CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

NL-339 (r. 82/08)

Canada

0-315-19365-4

National Library of Canada | Bibliothèque nationale du Canada

Canadian Theses Division | Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

67273

FACULTY OF GRADUATE STUDIES AND RESEARCH
14545 MAR 29 84
THE UNIVERSITY OF ALBERTA

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

KAREMPUDI VIJAYASUNDARA RAMARAO

Date of Birth — Date de naissance	Country of Birth — Lieu de naissance
25-7-54	INDIA

Permanent Address — Résidence fixe

VALAPARLA (PO), PRAKASAM (DT), A.P., INDIA

Title of Thesis — Titre de la thèse

RESILIENT DISTRIBUTED DATABASE SYSTEMS

University — Université

University of Alberta

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

Ph. D

Year this degree conferred — Année d'obtention de ce grade	Name of Supervisor — Nom du directeur de thèse
Spring 84	Francis Y. Chin

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date	Signature
21.3.84	KVS Ramarao

NL-91 (4/77)

THE UNIVERSITY OF ALBERTA

RESILIENT DISTRIBUTED DATABASE SYSTEMS

by

Karempudi V.S. Ramarao

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF Doctor of Philosophy

Department of Computing Science

EDMONTON, ALBERTA

Spring, 1984

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR Karempudi V.S. Ramarao

TITLE OF THESIS RESILIENT DISTRIBUTED DATABASE SYSTEMS

DEGREE FOR WHICH THESIS WAS PRESENTED Doctor of Philosophy

YEAR THIS DEGREE GRANTED 1984

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(SIGNED) ...K.V.S.Ramarao.....

PERMANENT ADDRESS:

....VALAPARLA (P.O)...
....PRAKASAM (Dt.)...
....A.P.) INDIA..

DATED ....16 Feb.....1984

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled RESILIENT DISTRIBUTED DATABASE SYSTEMS submitted by Karempudi V.S. Ramarao in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

.....

Supervisor

.....

.....

.....

.....

.....

External Examiner

Date.....

ABSTRACT

Database Systems are expected to guarantee the consistency of the data files in spite of any failures. One might simply construct systems that preserve consistency when there are no failures and dictate that no processing can be done when a failure occurs. This simple-minded solution is not acceptable in a distributed environment for the obvious reasons. We want the data at sites unaffected by the failure to be available to access and work on. In this thesis, we address this problem of maximizing the availability of distributed databases in presence of failures. We consider two classes of failures - simple (clean) site failures and network partitioning.

Transactions defined as atomic operations are the basic tools to guarantee the consistency. In a distributed environment, the sites participating in a transaction need to cooperate to decide whether the transaction can be completed or is to be aborted. To preserve consistency, they all need to take the same action. Protocols to achieve this goal are known as *commit* protocols and our concern is to design *nonblocking* commit protocols which can terminate all transactions incomplete at the time of a failure. Clearly, the availability can be maximized when such a protocol is followed. If it is not possible to find a nonblocking commit protocol, we would like to find protocols which maximize the availability.

It is known that there is no commit protocol nonblocking to (arbitrary) network partitioning. In this case, we introduce the

notion of non-trivial termination which is slightly weaker than the nonblocking requirement. We characterize the commit protocols which allow the non-trivial termination and show that one can have protocols that perform satisfactorily in presence of site failures as well as network partitioning. In fact, we present a fundamental relation between these two classes of failures, that the nonblocking problem for site failures is equivalent to the non-trivial termination problem for partitioning in a very strong sense. A simple commit protocol which allows non-trivial termination is studied at depth to obtain termination protocols optimal under certain practical measures that reflect the availability of the databases. The measures considered are, a) the expected number of sites that wait, and, b) the expected number of components that wait. Initially, we study this problem in a slightly restricted environment and later generalize to a case where different partitions could have different probabilities of occurrence and the specific properties of the protocol are also taken into account. But, we present the surprising result that the optimal protocols in most cases do not depend on this statistical information - showing that these protocols can be universally utilyzed.

As a tool for deriving the existence results and lower bounds on the message rounds for various classes of protocols, we develop an information-oriented model for distributed transaction-execution, which is extended to express protocols as well. Predicate Calculus is used to formally express the

statements in this model. Foundation for this model is the fact that any transaction-execution can be considered as an ordered sequence of three actions - initiation, decision-making and completion.

We study the special case of read-only transactions and present protocols nonblocking to both site and network failures. We introduce a *bounded-failure* model for site failures where not more than a fixed number of sites can fail simultaneously. Under this model, it is shown that the availability can be greatly improved in conjunction with site failures. Finally, we explore the recovery aspects of failed sites. Here, we study the possible recovery strategies and characterize the commit protocols that allow these strategies. We show one of them, where a site can recover after communicating to any operational site (that has participated in the transactions incomplete at the recovering site), to be a superior strategy. The relation between commit, termination and recovery protocols is also discussed.

Acknowledgements

I would like to express my sincere gratitude towards Dr. Francis Chin, my thesis supervisor, whose enthusiasm, encouragement and patience have been of a very high calibre. I would like to thank the members of my Examination Committee, Drs. Lenhart Schubert, Tony Marsland, W.W. Armstrong, C.A. Liu, and Tiko Kameda, for their helpful suggestions. I am thankful to Dr. Kameda for his thorough study of my thesis, which has made it more presentable, and for his several important suggestions.

Many people have helped to make my two-year stay at the University of Alberta an unforgettable experience. Firstly, I would like to express my appreciation for my wife, Rama, for her patience and support. Drs. Krishna, Rao, Srinivas, Prasad and Mr. Subba Rao need special mention for their help and friendly assistance on several occassions.

Table of Contents

1. Introduction 1

   1.1 Motivation 1

   1.2 Some Basics 2

   1.3 Introduction to the Area of interest 4

   1.4 Overview of our work 9

2. Model of the Environment 14

   2.1 Introduction 15

   2.2 The Distributed System 15

   2.3 The Database System 16

   2.4 Models for Distributed Transactions 18

   2.5 Models for Decision Making 21

   2.6 Protocols for Transaction-execution 22

   2.7 An Example 25

   2.8 Problems investigated in this study 30

   2.9 A formal model for protocols 31

3. An Information-based Model for Protocols 34

   3.1 Introduction 34

   3.2 A global model for transaction-execution 36

   3.3 Optimal nonblocking commit protocols 44

   3.4 Correspondence with the FSA model 57

   3.5 Read-only Transactions 58

   3.6 Summary 62

4. Site failures and Recovery 63
4.1 Introduction 63
4.2 Nonblocking protocols for bounded-failure networks 64
4.3 Failure Recovery 69
5. Network Partitioning - Preliminaries 75
5.1 Introduction 75
5.2 Problem Specification 76
5.3 Two Approaches to the problem 76
5.4 The structure of FSA 78
5.5 Termination Protocols 80
6. Optimal Termination Protocols for Partitioning - I 88
6.1 Introduction 88
6.2 Performance Measures 89
6.3 Decentralized Termination Protocols 90
6.4 Centralized Termination Protocols 101
7. Optimal Termination Protocols for Partitioning - II 107
7.1 Introduction 107
7.2 Analysis of the commit protocols 108
7.3 Partitions and probabilities 113
7.4 Measures for TPs 114
7.5 Decentralized TPs 115
7.6 Centralized TPs 122
7.7 Selection of the coordinator 125
7.8 Conclusion 126

8. Conclusions 128
8.1 Introduction 128
8.2 Resiliency of Distributed Databases - the past 128
8.3 Contributions of this thesis 131
8.4 Dealing with the other transaction models 137
8.5 Suggested future directions 139
REFERENCES 143
APPENDIX 151

Chapter 1 : Introduction

Contents

1.1. Motivation

1.2. Some Basics

1.2.1 Reliability of Databases

1.3. Introduction to the Area of interest

1.3.1 Fault-Tolerance

1.3.1.1 Maximization of Availability

1.3.2 Another Approach

1.4. Overview of our work

1.1. Motivation

'Have your own computer' - is the slogan of the day.

The rapidly changing world of hi-tech presents a picture of the future that was no more than science fiction a few years ago. A society in which people work, communicate and depend completely on computers is not far away. With individuals possessing personal computers which can do lot more than today's primitive personal computers, and communicating with each other on communication networks, it would be a world of Distributed Computing on a grand scale. Though the hardware tools towards this goal are becoming readily available at affordable prices, the achievement on the software front has been less than spectacular. Distributed Computing has brought with it several

problems that are entirely new and in many cases more complex than in a centralized environment.

User expectations of a distributed system are also far more ambitious - sharing enormous volumes of geographically distributed information, easy programmability, fast system responses, transparency to technical details like the location of the information, and, above all, a high degree of reliability being a few among them. Each of these requirements opens up a number of problems for the system designer to solve.

In this thesis, we address the reliability aspects of distributed systems. It is well known that in a distributed system, the probability that *some* component has failed is relatively high at any point of time. On the other hand, the probabilty that some component is *operational* is also very high. The reliability problem for distributed hardware systems has been extensively studied for several years. Here, we consider only the software systems and in particular, questions relating to database systems. Though our solutions are applicable to all distributed software systems, they are most relevant in the context of databases.

1.2. Some Basics

In what follows, we briefly introduce the reliability problem of distributed database systems.

A database can be considered as a collection of *entities*, an entity being a variable together with a set of values from its

domain. There are certain *consistency constraints* restricting the values a variable can take and relating the specific values of different variables. These constraints reflect the characteristics of the physical entities that are being modeled in the database. Thus, we require that these constraints always hold true in the database. As an example, let us consider a banking environment with a simple database : each 'record' consists of a transaction of the bank and the balance of the account affected by the transaction. Thus, the record of a transaction consists of the account number, debit/credit amount and balance. Sample consistency constraints are, a) all balances should be nonnegative, b) sum of the debits, subtracted from the sum of credits, should equal the balance for each account, and the like.

An instance of a database is said to be *consistent* if all the consistency constraints are satisfied in that instance. As these constraints reflect the physical environment, every instance of a database should be a consistent instance. We assume that any process accessing a database guarantees that the instance of the database obtained by completely operating that process is consistent, provided the initial instance it had acted on was consistent. We further require that the effects of a process on the database are either complete (ie., the process is **committed**) or null (ie., the process is **aborted**), since the consistency is not guaranteed otherwise. Such **atomic** processes are called **transactions** [Lampson 78]. Clearly, this is the first step towards ensuring that all instances are consistent. If

processes accessing the database act on it sequentially, then the consistency is trivially guaranteed since each of them individually guarantees it. Concurrent information sharing is the basic purpose of a database and should be provided for. On the other hand, it is easy to see that arbitrary concurrency does not ensure consistency.

The problem of achieving concurrency among the transactions while guaranteeing the consistency is known as the *concurrency control* problem for databases and has been extensively studied in the literature [BG 81, BG 82, EGLT 76, Gray 78, Kohler 80, Lampson 78, TGGL 82, Thomas 79]. We do not discuss this problem here.

1.2,1 Reliability of Databases: We say a centralized database is *reliable* if its consistency is guaranteed in spite of any combination of possible failures in the system. Since the distributed systems involve a number of local databases, we require an additional feature in that case: the database should be *available* in spite of $r \leq k$ simulataneously failed components, for some reasonably large k. Thus, we want to ensure that the system is not sensitive to the failure of a few components. Otherwise, the availability of the system will be unsatisfactory, counter to the philosophy of distributed computing. In the event of failures, we want the database to be available at least for some transactions, probably exhibiting a lower level of performance, rather than being totally unavailable.

1.3. Introduction to the area of interest

Most articles in the literature have dealt with the reliability problem of distributed databases in a divide-and-conquer fashion. The approach taken is -
a) solve the concurrency control problem in a no-failure environment,
b) devise means of consistently updating the database in presence of failures (*failure atomicity*), assuming the existence of concurrency control mechanisms, and, finally,
c) maximize the availability of the system in presence of failures.

We do not deal with the concurrency control problem here. The interested reader is referred to any of the several nice survey articles by Bernstein et al [BG 81, BG 82].

1.3.1 Fault-tolerance

Fault-tolerance in central systems is easier to handle than in distributed systems. The possible faults here are limited to : a) processor/main memory failure, b) secondary storage failures (eg., disk head crash), and c) software failures such as unhandled exceptions. The atomicity of transactions can be guaranteed as follows: the effects of the transaction are first recorded temporarily and are made permanent only when the transaction can no longer be aborted. If the transaction is to be aborted, the temporary record is then discarded. We can see that this solution works in presence of any of the above failures. The key point is that the effects are not made permanent unless it is guaranteed that an abort is impossible.

There are several possible failures in distributed systems: a) communication failures, e.g., missing/duplicated messages and link failures, b) node failures, e.g., site failures and processor malfunctioning, and, c) network partitioning which arises either due to site failures, link failures or a combination of both. Most of these failures can critically influence the transaction atomicity. The simplest is the link failure which poses no problem for communication among the sites as long as the network is connected. The problem of missing messages can be solved by repetitive transmission of a message until it is acknowledged. This process itself can give rise to duplicate messages and several simple solutions are available to deal with duplicated messages. For instance, each site records the identity of the latest message received from each of the other sites so that the earlier messages can be considered duplicates. A number of slightly more complicated and robust solutions also exist.

The problem of processor malfunctioning has been extensively studied [DR 82, DS 82, DS 82a, DS 82b, DS 82c, LSP 82, PSL 80]. Cases of authenticated and forgeable messages have also been explored in some of those articles. Optimal decentralized algorithms are presented for a number of correctly functioning processors to reach consensus in the presence of several malfunctioning processors. A special case is when some sites fail while some others are malfunctioning.

When no processor malfunctioning is possible, the solutions of the general problem mentioned above are not very attractive in the number of messages and message rounds. Hence, this limiting case is also well-studied in the literature.

Network partitioning has been found to be the toughest of all to deal with [SS 81]. The existing systems have assumed that the network partitioning is a *catastrophy* and left it unhandled [HS 80].

1.3.1.1 Maximization of Availability

As we have seen, transaction atomicity requires communication among the participating sites. As a consequence, when some components of the system fail, it may not be possible for the operational components to proceed until some of the failed components recover. For instance, if a transaction is accessing a large fraction of the database entities and a number of sites are participating in it, the failure of some of them at a critical stage might require that the operational sites wait for some of the failed sites to recover. In this case, most of the database entities are not available for any other transaction unless this transaction is completed either by committing or by aborting it, which can happen only after the recovery of some of the failed sites. This undesirable situation should be avoided. We wish to see that, irrespective of whatever components fail, the operational components can complete all the pending transactions and proceed with the other transactions. A protocol with this property is known as a *nonblocking protocol*. At the

same time, we should ensure that the action of the operational components cannot be inconsistent with the failed ones since the commit and abort actions are *irreversible*. Such protocols are known as *nonblocking commit protocols*.

The problem of finding nonblocking commit protocols has received attention only recently [Skeen 81, Skeen 82, SS 81]. Nonblocking commit protocols have been developed for the case when only site failures are allowed. On the other hand, it has been proved that there is no nonblocking commit protocol when network partitioning involving more than two groups of sites occurs [SS 81].

1.3.2 Another approach: Some investigators have taken another interesting approach to the problem of distributed database reliability : they tried to combine the problems of concurrency control and failure atomicity and find solutions for this combined problem [BL 82, Gifford 79, LB 82, Skeen 82a, Thomas 79]. These techniques are expected to work irrespective of whether there are any failures or not. Typically, the failures anticipated are network partitioning and site failures. A useful consequence of this approach is that the occurrence of these failures need not be detected. Unfortunately, such a technique should anticipate the worst failure and hence tends to be pessimistic.

Thomas has developed an interesting solution for database updates utilizing the time-stamps, known as *majority consensus* [Thomas 79]. Typically, all participating sites vote, indicating

their willingness to commit or abort the transaction. The decision of a majority of them is implemented. Thus, independent of any failures, the transaction execution continues and the consistency is guaranteed as long as a majority agreement is possible. This approach is very useful when dealing with multi-copy updates. Later, Bruitweiser et al [BL 82, LB 82] also have used a similar idea. Skeen has generalized the majority concept to quorums and applied it in the failure atomicity problem [Skeen 82a].

1.4. Overview of our work

We address the problem of maximizing the availability of distributed databases in presence of site and network failures. Preservation of the consistency of the database is considered as of utmost importance and hence cannot be given up even temporarily. Atomic transactions are convinient tools to preserve consistency after acting on a consistent instance of the database, so that, guaranteeing the atomic implementation of transactions in presence of failures implies the consistency. Hence, we are interested in protocols that guarantee the *all or none* effect of transactions in distributed databases. Since all transactions need not be committed, we need means of deriving the final decision on the mode of completion of a transaction, which is obtained by all the sites involved and implemented in a consistent fashion. We study two models for this - the centralized model where a specific site decides and the decentralized model where each site votes according to its

preference. We propose an information-oriented model for the execution of a distributed transaction, so that the protocols that govern the completion of transactions can also be similarly modeled. The protocols can then be expressed using first-order predicates depicting the information possessed by the sites participating in the protocol. We can express various properties of protocols by saying that a certain class of predicates are or are not among those realized under a given protocol. We are specifically interested in the *nonblocking* property of protocols which enhances the availability of the database. We derive the necessary and sufficient conditions that a commit protocol is nonblocking to site failures, under the two models for reaching a decision. We show that, under the central model, any commit protocol allowing aborts is nonblocking. This interesting fact can be attributed primarily to the realization that, a) if the supervisor is operational, then the failures of other sites can be easily taken care of , and, b) when the supervisor is down, the operational sites can complete depending on whether all operational sites have received the transaction or not. For the decentralized model, we show that a commit protocol is nonblocking if and only if a site is not allowed to commit unless all sites know that the decision of all other sites is also commit. We define the notion of a step to model the time duration of a protocol and derive lower bounds for the different classes of commit protocols. The nonblocking commit protocols, in the decentralized model, are shown to be inherently more expensive than the blocking protocols.

It is known that no commit protocol is nonblocking to arbitrary partitionings of a network. We introduce the notion of *non-trivial termination* which is slightly weaker than the nonblocking requirement but increases the availability appreciably. We prove that there is no commit protocol under the central decision model, which allows non-trivial termination. This implies that the protocols employing the central decision model are totally ineffective in dealing with partitioning. On the other hand, we have a positive result when the decision is decentralized, which says that a large class of protocols allow non-trivial termination. In fact, we prove an interesting relation between the site failures and the network partitioning, showing that a protocol allows non-trivial termination if and only if it is nonblocking to site failures. This result thus brings out the specific subproblem of the nonblocking problem for partitioning which is equivalent to the nonblocking problem for site failures.

It is well-known that a large number of transactions on databases are read-only in nature. We consider this special class of transactions and present simple, fast protocols which are nonblocking to both site and network failures. We show how these protocols can be easily extended to deal with the hierarchical and nested models of distributed transactions. Maximum availability is achieved in this case by observing that participating sites can unilaterally commit or abort such transactions.

Site failures studied till now assume that any number of sites can fail simultaneously. We propose a model where only a bounded number of sites can simultaneously fail. Using the notion of backups, the termination and recovery aspects are greately improved under this model.

Sites recovering from failures need to complete the transactions incomplete at the time of their failure. To achieve higher availability, we require that they can do so as soon as they can. Hence, we study the possible recovery strategies that could be used by the recovering sites while guaranteeing the consistency. We relate the commit, termination and recovery aspects of protocols, showing one of the recovery strategies to be the most appropriate in conjunction with nonblocking protocols.

Having seen that there is a large class of commit protocols under the decentralized model of decision, which allow non-trivial termination, we probe further to find what best we can do to improve the database availability. Specifically, we look for termination protocols optimal under certain measures of practical interest. First, we consider the slightly simplistic environment where all partitionings are equally probable. The measures we consider are, a) the number of *components* (groups, together with their states), and, b) the number of sites, that wait under a termination protocol. We develop the theory of termination protocols and derive the optimal ones. Interestingly, we show that simple quorum-based termination

protocols outperform most of the other protocols. We study both the centralized and decentralized protocols and derive the optimal protocols. Finally, we generalize our environment by considering that different partitions could occur with different probabilities and deriving the probabilities with which different components can occur under a given protocol. We produce optimal protocols under the generalized measures, while providing the surprising result that the statistical information hardly makes any difference, in that, most of the protocols optimal in the simplistic environment still remain optimal. This result can be primarily attributed to the fact that the probability with which a set of sites are all in a specific state is very large relative to the probability with which they could be in different states. But, these results have the far-reaching consequence that the optimal protocols can be used in any environment with equal effectiveness.

Chapter 2 : Model of the Environment

Contents

2.1. Introduction

2.2. The Distributed System

2.3. The Database System

2.3.1 User interaction - Transactions

2.4. Models for Distributed Transactions

2.4.1 Centrally-controlled transaction

2.4.2 Relaxed central-control transaction

2.4.3 Hierarchical transaction

2.4.4 Nested transaction

2.5. Models for Decision Making

2.6. Protocols for transaction execution

2.6.1 Classification of commit protocols

2.6.1.1 Centralized commit protocols

2.6.1.2 Decentralized commit protocols

2.6.1.3 Desirable properties of commit protocols

2.7. An Example

2.7.1 The Two-phase commit protocol

2.7.2 Termination protocols

2.7.3 Recovery protocols

2.8. Problems investigated in this study

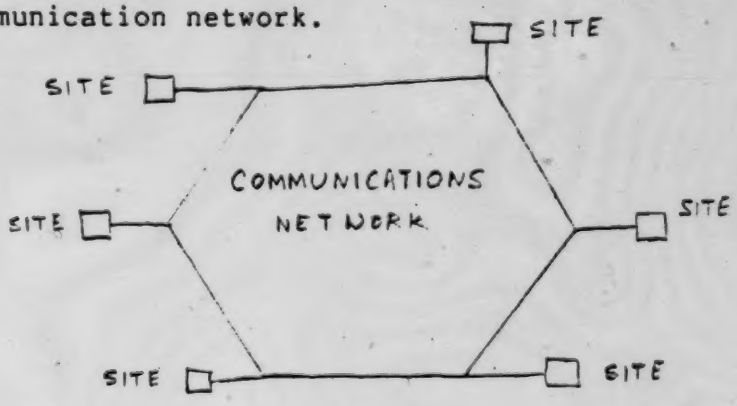
2.9. A formal model for the protocols

2.1. Introduction

In this chapter, we formally describe our working environment. Our notion of a distributed system is presented through the definition of its components and their interactions. The distributed database environment and the user interaction with it is then described. The important notion of transactions and several models of distributed transactions are presented. Some details of transaction execution and the notion of commit protocols are given, and, a classification of commit protocols and a detailed description of a well-known commit protocol is presented.

2.2. The Distributed System

A distributed system consists of two components - nodes and the communication network.



Any two nodes interact via *messages*, a message being a bitstring transmitted through the communication network. A node submits a message accompanied by a valid destination address to the network. We do not require the network to tell the node whether the message has been delivered to the destination or not, or, to

care whether the destination node is operational or not. But, we require that the network delivers messages from any node i to any other node j in the order they are generated. However, there is no order among messages generated at different nodes.

A node is considered to have failed if its processor fails. The components of a network include a number of communication links. A link is said to have failed if no bitstrings can be transmitted over that link. Link failure is the only way a network can fail.

We abstract a node as consisting of a processor and two kinds of storage, the volatile and the permanent. The processor reads its instructions from the volatile storage and performs I/O with the volatile storage and/or the network. When the processor fails, the following things happen: a) the contents of the volatile storage are completely wiped out, and, b) all input/output activity of the processor comes to a complete stop. By definition, the permanent storage *never* fails.

2.3. The Database System

In a distributed database, the files are distributed among the nodes of the distributed system. A node can contain any number of files (including zero) and a file can be resident at any number of nodes. To ensure consistency of the database, it should be guaranteed that a) all copies of a given file are mutually consistent (if the operations on the database come to a halt, then all copies of the file are identical), and, b)

different files are (internally) consistent , meaning that the consistency constraints hold.

2.3.1 User Interaction - Transactions: The user interacts with the database by issuing commands that access the database. It is easy to see that individual commands issued need not guarantee the consistency of the database. As an example, let us consider the banking database mentioned in Chapter 1. Consider the command

Withdrawal(Acc-no) ← 1000.

After the execution of this command, the database is no longer consistent since the value of Balance(Acc-no) is unaltered while the Withdrawal(Acc-no) is modified.

This example shows that a sequence of database access commands is required to guarantee consistency. The sequence of commands which constitutes a *unit* for database access, is called a **transaction** [Lampson 78]. A transaction is said to be an **atomic** action on the database since partial transactions do not guarantee consistency. Thus, we require that the effects of a transaction on a database are either complete or void.

When a number of transactions access the database concurrently, mechanisms should be used which guarantee that the cumulative effect of all those transactions on the database entities is equivalent to an execution where individual transactions are run in isolation and in some sequence. This is known as the *serializability* problem for interleaved sequences of transaction subactions. Bernstein et al. give a good exposure to

this problem in the distributed environment [BG 81, BG 82]. There are several papers dealing with this problem for centralized databases [EGLT 76]. In this thesis, we do not concern ourselves with this aspect.

In a distributed environment, a transaction submitted at a node may require database entities stored at other nodes in the system, requiring a cooperative execution at a number of nodes. These other nodes will be referred to as the *participating sites* of that transaction. If a transaction is executed to completion so that its effects are permanently incorporated into the database, we say that the transaction is *committed*. If one of the participating sites cannot complete the transaction, i.e., *aborts* it, then all other sites should abort the transaction. Observe that inconsistencies may be introduced into the databases if some sites abort and some sites *commit* the transaction. There are several reasons why a transaction cannot be completed - deadlocks and hardware failures, for instance. In our example with a banking database, a transaction that tries to withdraw more than the current balance cannot be committed. Thus, a transaction initiated in a system is permitted to end only in two possible ways - either it is committed or aborted.

2.4. Models for Distributed Transactions

Depending on how the overall execution of a transaction is controlled, we have a number of models for distributed transactions.

2.4.1 Centrally-controlled transaction [Gray 78]: There is a designated site, called *supervisor* (or *central site*), to carry out all the control functions corresponding to a transaction execution, irrespective of where a transaction is submitted. The supervisor analyzes the transaction and determines the participating sites. It prepares a detailed list of all the actions to be taken at each of the sites and sends out the appropriate commands to them. The participating sites carry out these commands.

This model is quite simple and hence makes it easy to solve various associated problems like concurrency control, deadlock handling etc. On the other hand, the entire system is threatened to be paralyzed if the supervisor fails. Further, this model does not effectively support heterogeneous distributed databases, where different sites may have different kinds of database management systems locally.

2.4.2 Relaxed central-control transaction [Gray 78]: In the above model, failure of the supervisor is disastrous. We can easily relax this situation by designating a number of sites as supervisors so that each of them manages the transactions originated at a specific subset of sites. This modification reduces the sensitivity of the system to individual site failures. SDD-1 is an existing system that employs this model [HS 80].

2.4.3 Hierarchical transaction [Gray 78]: In this model, the transaction originated at a site, called the *root-transaction*,

creates subtransactions at (possibly) some other sites. Each subtransaction can in turn create its own subtransactions, so that this process continues to any depth. At any level in this hierarchy, a transaction is aware of the process that has generated it (the parent) and the process(es) it has generated (sons), if any. As far as its effects at that node are concerned, a subtransaction at a node is in no way different from a transaction originated at that node. If the execution of a subtransaction at a node requires the services of several systems at that node (like file management system), it would generate subtransactions to deal with each of these services. Thus, this model supports heterogeneous databases.

A transaction at any level in the hierarchy can decide to commit only if all its subtransactions decide to do so and it should abort if any one of them aborts. Similarly, if a transaction aborts, all its subtransactions abort. Thus, if *any* transaction at any level in the hierarchy aborts, the whole hierarchy should abort.

2.4.4 Nested transaction [Moss 81, Moss 82, Reed 83, RS 82]: Observe that, in the hierarchical model, a transaction such that two of its subtransactions compete for resources at the same site can never be committed. Since no transaction in the hierarchy knows the details of all the other (sub)transactions in the hierarchy, there is no way of ensuring that such a situation does not arise. Thus, the hierarchical model needs enhancements to be general enough to deal with such cases. We need some scheme for

synchronization among the transactions of the same hierarchy which makes the effects of certain transactions visible to others before the root-transaction decides to commit. The hierarchical model with such a facility is known as the *nested* transaction model.

We choose the Relaxed central-control model. Solutions in this model can be directly extended to the other models. This model is the simplest to deal with and gives deep insights into the problem of failure handling. We assume that each site in the system is the supervisor for all transactions originated at that site [BG 81].

2.5. Models for Decision Making

In each of the transaction models, it is imperative that a single site first receives the transaction request from the user. This site in turn may request some other sites to participate. Since a transaction can eventually be either committed or aborted, there are two possible ways in which that decision can be made - a) each participating site decides on its part of the transaction and the overall decision is derived from these individual decisions, or, b) a designated set of sites make the decision and the others obey it. We call these models *Decentralized Decision* and *Centralized Decision* models respectively. A limiting case of the latter model is the one in which the designated set contains a single site. In this case, the designated site decides whether to commit or to abort the transaction in view of the database consistency. In our study,

we consider both of these models. For the second model, we consider the limiting case of a single designated site.

2.6. Protocols for Transaction Execution

Let us recall that a distributed transaction can be simultaneously executed at a number of sites in the system, each site executing a part of it. Since no site can always correctly 'guess' the status of the other sites, communication among the sites is required, possibly interleaved by some local actions, to ensure that atomicity is guaranteed at all times. Such an action-communication pattern followed by the sites participating in a transaction is known as a *protocol*, and a protocol that always guarantees atomicity of transactions in the absence of any failures is called a **commit protocol** [AD 76]. When there are failures, the commit protocol by itself may not be sufficient to handle the exceptions. Some special protocol may be required to complete the processing of a transaction. These special protocols are known as **termination protocols** [SS 81]. Similarly, after a failure has been repaired, the components of the system recovering from the failure may have to take special actions to deal with the transactions incomplete at the time of the failure. Such protocols are called **recovery protocols** [SS 81]. Thus, a commit protocol is normally associated with some termination protocols and some recovery protocols.

2.6.1 Classification of commit protocols:

Consider the decentralized decision model. In our transaction model, a transaction is submitted at one of the sites which acts as the supervisor. Once a participating site receives a participation request, it executes its part of the transaction and makes a local decision. All the participating sites now need to communicate and cooperate in order to reach a final decision. Depending on this communication pattern, we can have different classes of commit protocols.

2.6.1.1 Centralized commit protocols [AD 76]: The simplest way of achieving the above goal is to let one of the sites pool together all the local decisions, derive the overall decision and pass it on to all the others. This solution clearly requires the smallest number of messages. Another centralized way of completing the transaction is hierarchical. Leaf nodes in the hierarchy initiate by sending their decisions to their parents. A parent sends out commit or abort message to its parent depending on its own decision and of its sons. The root of the hierarchy, after receiving the messages from its sons, makes the final decision and sends it out to its sons, propagating it down the hierarchy.

2.6.1.2 Decentralized commit protocols [SS 81]: In the above class of protocols, the role of all sites is not the same - there is a master site and all others are slaves. We can make all sites equal in their status so that there is no master-slave relationship among the sites. Now, any site can communicate with any other site. Protocols obtained this way are decentralized -

each site sends its decision to all other sites and hence receives the decision from all other sites. It can simply determine the final decision and implement it.

2.6.1.3 Desirable properties of commit protocols: The notion of commit protocols is introduced to guarantee the atomicity of distributed transactions in the absence of failures. We would like to find commit protocols optimized in the following measures : a) number of messages transmitted, b) amount of parallelism involved, (reflected by the total amount of time for transaction completion) and c) number of writes onto the permanent storage.

Though we have considered commit protocols in failure-free environments, no system is failure-free. Failures can occur any time during a transaction execution and we require that the commit protocol does not break down in the event of some failures. Hence, we desire the following additional properties for commit protocols:

a) If a particular kind of failure has occurred (say, some sites have failed), the operational components of the system need not wait for any of the failures to be repaired. In our terminology, this implies that there are termination protocols (each possibly associated with a class of failures), that ensure the completion of incomplete transactions at all the operational sites. A commit protocol is *nonblocking* to a class of failures if there is a termination protocol of the given protocol associated with that class of failures and has the above property. If there is no

nonblocking commit protocol for a class, we would like to consider those protocols that maximize the the number of sites that do not wait.

b) We further require that the commit protocol facilitates the recovery of components that have failed earlier. Ideally, we would like to see that any component can recover on its own, i.e., it need not communicate with any other component to recover completely (*independent recovery* [SS 81]). If it is not possible, we wish that the number of messages passed and the total time for recovery are minimized.

2.7. An Example

As an example, we present a well-known protocol, the *Two-phase commit protocol* [Gray 78]. We analyze its costs in terms of the various measures we have listed earlier. We then give termination protocols to deal with the failure of some of the participating sites and compute their costs. Finally, we give recovery protocols associated with these termination protocols.

2.7.1 The Two-phase commit protocol [Gray 78]: Assume that a transaction is presented at site i. We call i the *supervisor* for this transaction. Site i sends the participation requests for all participating sites. After receiving such a request, the participating sites process the request and determine whether they can commit the transaction or not. Such a site first records the transaction identity, the supervisor's identity and

its readiness to commit or abort, into the permanent storage. It then sends an 'yes' or a 'no' message to the supervisor accordingly. Having received messages from all the participating sites, the supervisor determines the final decision. It first records the transaction identity, identities of all participating sites and the final decision, into its permanent storage. It then sends out 'commit' or 'abort' messages to all the participating sites depending on the decision. A participating site, after receiving the decision, records the transaction identity and the final decision in its permanent storage and then completes the execution of that transaction by taking all the required actions like releasing the resources. Then, it sends out an 'ack' to site i and removes all the information on that transaction from its volatile storage, in effect 'forgetting' about that transaction. Site i, after receiving all the 'ack's, records the completion of that transaction in its permanent storage. When the site eventually records it, all information on that transaction is removed from its volatile storage.

We see that this protocol has two phases apart from the step in which participation requests are distributed. The participating sites send their decisions to site i in the first phase and receive the final decision and send acks in the second. The first phase consists of n messages if there are n participating sites and these messages are sent asynchronously. Assume t(j) is the local processing time at site j and d(j) is the time taken for a message between sites i and j. Then the total time taken in till the completion of the first phase is

max{t(j)+2d(j)} (This includes the initialization also).

The second phase consists of 2n messages. If the network has broadcasting facility, n of them can be sent simultaneously. Otherwise, these n messages are to be sent sequentially. The other n are sent asynchronously in both cases. If c(j) is the time taken by j to implement the transaction completion, this phase takes max{c(j)+2d(j)}, assuming that i takes negligibly small time to make the final decision. Each of the sites (including i) writes one record on its permanent storage in this phase. Site i writes one more record later.

Thus, the Two-phase protocol involves 3n messages, a time span of max{t(j)+2d(j)}+max{c(j)+2d(j)} and (2n+1) records written into the permanent storages. One additional record is eventually written by the supervisor to mark the end of the transaction.

2.7.2 Termination protocols: Let us assume that there are some site failures during the execution of a transaction.

Case 1. When the supervisor has not failed: If a failure has occurred before a site has sent its decision, then the supervisor can assume that the failed site wants to abort the transaction and direct others to abort it. If the site fails after sending its decision, then the supervisor can ignore the failure and implement the decision it has made.

Case 2. When the supervisor fails: There are two possible situations in this case. If the participating sites cannot communicate among themselves, then they all have to wait until

the supervisor recovers. When the supervisor recovers, they send queries on the decision for that transaction. If they can communicate, they elect a new supervisor which collects the local decisions of the surviving sites. If one of them is abort, it directs all of them to abort. If one of the surviving sites has already received a 'commit' or 'abort' message from the old supervisor, then that decision is carried out. If all the surviving sites decided to commit and no site has received the decision from the old supervisor, the new supervisor can direct to commit only if all participating sites are operational (i.e., only the site i has failed.) Otherwise, they all wait.

Thus, if the participating sites cannot communicate among themselves, they wait if the supervisor fails and thus need no extra messages. But, an indefinite amount of time might elapse before the supervisor recovers. If they can communicate, they have to elect a new supervisor, which requires O(n logn) messages [Garcia 82]. Even then, they may have to wait as shown above. In both the cases, there are situations where the operational sites wait for some failed sites to recover. In fact, it has been proved [Skeen 81] that it is the case with *any* termination protocol of any version of the Two-phase protocol.

2.7.3 Recovery protocols: From the above discussions, it is not hard to see that recovery protocols depend both on the commit and termination protocols used. Corresponding to the two termination protocols above, we give recovery protocols here.

a) *When participating sites cannot communicate among themselves:*

By our definition, the volatile storage is erased in the event of a site failure. When a site recovers from failure, it first reads the permanent storage and extracts all the information on transactions that were not completed. Let us consider a participating site first, with a single record containing the transaction identity, supervisor's identity and the local decision. If the decision is an abort, it can complete by writing an 'abort' completion record and performing other actions. If it has recorded a 'commit' decision, then it cannot make a unilateral completion since some other site may have aborted. Thus, it sends a query to the supervisor of that transaction. On receiving the answer, it completes the transaction and sends back an 'ack'.

Now, let us consider the failure of the supervisor. Recall that it makes the first entry in the permanent storage only after receiving all local decisions. Hence, it may not have any information on some of the transactions it was coordinating. Since there is no possible communication among the participating sites, all participating sites should have been waiting for this site to come up and those of them whose decision was 'commit' now send queries to the supervisor. If it does not find any information on the queried transaction, it can safely direct them to abort. Assume it has a single record on a transaction. Then it sends that decision to all participating sites (that list is available in the record) and proceeds exactly as in the normal mode.

b) *When participating sites can communicate among themselves*: The

situation is the same as above for a participating site. The new supervisor makes sure that the decision it could take, if any, is passed on to the old supervisor as soon as the old supervisor recovers. If the new supervisor fails before the old one recovers, the old supervisor waits until it hears from the new supervisor. (Observe that it cannot query the new supervisor since its identity is not known.)

Thus, we see that, under both of the recovery protocols, recovering sites in general need to communicate with some other sites and may have to wait for some specific failed sites to recover in some situations. Typically, two messages are required for recovery and no extra actions are required. Time delays can be indefinite in some cases.

2.8. Problems investigated in this study

We study two classes of failures - site failures and network partitioning. We investigate the atomicity of distributed transactions under these failures. Specifically, we consider the problem of designing protocols that guarantee atomicity in the presence of these failures, and have the desirable properties listed in Section 2.6.1.3. We investigate the existence of nonblocking commit protocols. In the cases where we have proved the non-existence of such protocols, we define several measures that represent the maximization of database availability and look for protocols optimal under these measures. The transaction model we use is the relaxed central-control model. We study two classes of commit protocols - centralized and decentralized and

consider two modes of decision making - centralized and decentralized.

2.9. A formal model for the protocols

The model for protocols discussed here due to Skeen [SS 81] and is based on the finite state automata. A protocol, to be followed by each of the participating sites, can be modeled by a finite state automaton. Thus, the collection of the automata at all participating sites represents the execution of a distributed transaction. The communication network is the common input/output tape shared by all sites. A site reads the messages addressed to it and writes messages with valid destination addresses. A finite state automaton (FSA) in a state reads a set of messages from the tape, possibly does certain internal computation and outputs a set of messages (possibly null) and changes its state. Output and state change are considered instantaneous, but the internal computation can take a finite amount of time.

FSA at different sites need not be identical. But, all the FSA share certain important restrictions - a) the final states of an FSA are divided into two disjoint subsets denoted as Com and Ab, known as 'commit' and 'abort' states, b) FSA is acyclic, c) FSA is nondeterministic, and d) there are no transitions from a state in Com (Ab) to a state not in Com (Ab). As an example, we represent the Two-phase commit protocol by FSA.

Supervisor FSA

Participating site FSA

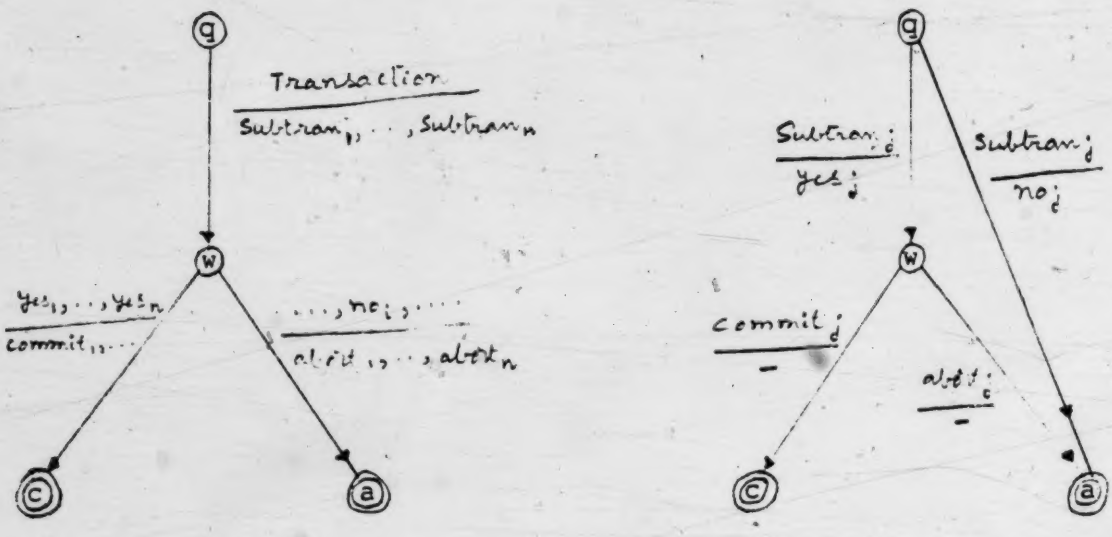


Fig. 1 : Two-phase commit protocol

Let us notice that there are severe constraints on the states that could occur concurrently with any given state. For instance, if a participating site is in state q, no other participating site can be in state c. Similarly, if the supervisor is in state q, no other site can be in w. In general, a state is *concurrent* to the states adjacent to it in the state diagrams, for this protocol.

The site failures are modeled through the traditional 'time-out' technique. In terms of FSA, a failure is considered

as a transition from the state before failure to a state it would enter into, after the failure recovery. A failing site reads all the outstanding messages destined to it, does no internal computation and writes 'time-out' messages to all other sites. Thus, failure can also be expressed consistently with the normal operations.

Using FSA, which depict the behaviour of individual sites, global behaviour on the transaction can be represented as a global state. A global state is the vector containing all local states and all outstanding messages in the network. Though the concept of global time, and hence the global state, do not have physical realization, the constraints on the possible concurrency among the states make the idea of (logical) global state useful in studying the properties of commit protocols. Several interesting results have been derived by Skeen using this model [Skeen 81, Skeen 81a, Skeen 82a, SS 81]. We present some more details of this model in later chapters. The interested reader is referred to [Skeen 82].

Chapter 3 : An Information-based Model for Protocols

Contents

3.1. Introduction

3.2. A global model for transaction execution

3.2.1 Nonblocking commit protocols

3.3. Optimal nonblocking commit protocols

3.3.1 Delayed-commit protocols under the centralized decision

3.3.2 Delayed-commit protocols under decentralized decision

3.3.2.1 Fundamental relation among failures

3.4. Correspondence with the FSA model

3.5. Read-only Transactions

3.6. Summary

3.1. Introduction

In Section 2.9, we have introduced a model for commit protocols, proposed by Skeen. Though this model is quite general, there are certain aspects that are not easily expressible. For instance, the concept of *back-up sites*[1] is one of them. Introducing a new FSA corresponding to a back-up would explosively increase the number of global states, and would require more complicated analysis, while the concept of back-ups

[1]This concept will be discussed in Chapter 4.

is intuitively simple. Another aspect that is not explored is the Centralized Decision model [Section 2.5]. In general, given the FSA of a protocol, it is not easy to understand how it works.

In this chapter, we propose a global information-based model, which represents a distributed transaction as a set of actions that transform a system satisfying a specific precondition into one that satisfies a specified post-condition. Even though some of the results derived from this model have appeared in other forms, our model gives the exact information required at each stage of the protocol during the execution of a transaction in an explicit fashion. Lower bounds on the number of message rounds can be easily derived under our model. In addition, this model is conceptually simpler and more attractive than the existing ones. We study the nonblocking commit protocols under the centralized and decentralized models of Decision, consider site failures and network partitioning and present a unified approach to the nonblocking properties of commit protocols. Several existence results on the nonblocking properties of commit protocols and an important relation among the two classes of failures will be presented.

3.2. A Global Model for Transaction Execution

A distributed transaction may have many subtransactions, one for each site. It is our intention that every site follows the same decision, i.e., either commit or abort its subtransaction. Without loss of generality, assume that there are n participating sites and that they are numbered 1 to n. For any site i, $1 \le i \le n$, we define the following predicates and functions.

Subtrans(i) - the subtransaction for site i is contained at one of the sites with which i can communicate.

Transaction(i) - site i is in possession of the subtransaction to be executed there.

Decision(i) indicates what site i knows about the global decision.
Decision : {1,2,...,n} → {'commit','abort','none'}
Decision(i)='none' means that site i has not obtained any global decision at that point of time.

Complete(i) indicates the mode of the transaction completion by site i.
Complete : {1,2,...,n} → {'commit','abort','none'}.
If its value is 'none', site i has not yet completed the transaction.

Recall that, since the 'commit'/'abort' of a transaction is *irreversible*, the participating sites should either all commit or all abort in order to maintain the consistency of the database.

Definition 3.1: A *Transaction-execution* is any action that is comprised of the following sub-actions, some of which can possibly be null:

Transaction-initiation is any action that transforms a system satisfying the precondition C_1: $(\forall i)$(Subtrans(i)), into the one satisfying the post-condition C_2: $(\forall j)$(Transaction(j)).

Reach-Decision is any action that transforms a system satisfying the precondition C_2 into the one satisfying the post-condition C_3: $(\exists k)$(Decision(k)≠'none').

Transaction-completion is any action that transforms a system satisfying the precondition C_3 into one that is satisfying the post-condition C_4: (∀k)(Complete(k)≠'none').

This abtraction of transaction-execution encompasses all the models of transactions. If the transaction is submitted at a site which can be assumed to be a central site, that site sends out the transaction components to the participating sites, realizing the transaction-initiation step of our abstraction. Also, our abstraction permits a distributed transaction to be simultaneously submitted at a number of sites. For reaching a global decision, we leave the Reach-decision step general enough to take care of the centralized and decentralized models of decision. In fact, any possible form of interaction among the sites to reach a global decision is equally well-expressed in this formalism. Same comments hold for the transaction-completion too. Moreover, since the specific forms of site interaction are not specified in our formalism, our model can represent the execution of a transaction under any protocol. This is in contrast to the other models where the distinction between the protocol followed and the transaction-execution process is blurred. Furthermore, our model is highly flexible by allowing different models at different stages of the transaction-execution. For instance, one can have a centrally-controlled model for transaction-initiation, a decentralized model for decision, a hierarchical model for reach-decision and a decentralized model for transaction-completion.

In the following, we study the consequences of all the different combinations of these three actions. Clearly, Transaction-initiation can never be null.

Definition 3.2: A *protocol* P is a set of rules for the above three actions and is called a *commit protocol* if, when there are no failures, we have
$(\forall j,k)$(Complete(k)=Complete(j)≠'none') at the end of transaction-completion, for any transaction-execution under P [AD 76, Gray 78].

We have established a natural correspondence between protocols and the distributed transaction-execution under them. The Two-phase commit protocol described earlier in Section 2.7, can be expressed as follows:

Transaction-initiation : relaxed central-control model
Reach-decision : centralized model
Transaction-completion : centralized
Model of decision : decentralized.

While proving results on protocols, we present the protocols at a slightly lower level by explicitly stating the rules of actions.

Notation: We use the mathematical implication symbol '→' in the following sense - A → B for any two predicates A,B iff, if A is true (at time t), then B is also true (at time $t_0 \geq t$).

For any commit protocol P and a distributed transaction, we have the following properties:

P1. $(\forall k)(\text{Complete}(k)=\text{'commit'} \rightarrow \text{Transaction}(k))$.

P2. $(\forall k)(\text{Complete}(k)=\text{'commit'} \rightarrow \text{Decision}(k)=\text{'commit'})$

P3. $(\exists k)(\text{Complete}(k)=\text{'commit'}) \rightarrow (\forall j)(\text{Decision}(j)\neq\text{'abort'})$
$(\exists k)(\text{Complete}(k)=\text{'abort'}) \rightarrow (\forall j)(\text{Decision}(j)\neq\text{'commit'})$

Intuitively, these properties assert that, 1) having its component of the transaction is a prerequisite for any site to commit a transaction, 2) if a site completes the transaction, it should have known the global decision, and 3) no site can commit if any participating site has a decision 'abort' and vice versa.

Notice that the above properties need not hold after there are failures, since a different protocol is invoked after a failure, whose actions would be different from those of the commit protocol.

3.2.1 Nonblocking commit protocols

The failure of a site has the following implications: a) processing at the site comes to a complete halt, and, b) no messages are generated by the site. Network partitioning occurs when the set of sites gets partitioned into a number of mutually disjoint subsets (called *groups*) such that a) sites within the same group can communicate with each other, and, b) sites in different groups cannot communicate. A group is *active* if the sites in it are operational. It is *inactive* otherwise. In our context, a useful piece of information two sites in the same

active group can share is the decision on a transaction in which they are participating. We formally express this as,

P4. $(\forall i \in S)(\text{Subtrans}(i) \rightarrow \text{Transaction}(i))$, and,

$(\exists j \in S)(\text{Decision}(j) \neq \text{'none'}) \rightarrow (\forall i \in S)(\text{Decision}(i) \neq \text{'none'})$,

for any group S.

One can view site failures also as partitioning - all operational sites together form the only active group and each of the failed sites forms an inactive group by itself. By contrast, in the event of a partitioning, there are more than one potentially active groups. In what follows, we view both these failures as two cases of a single type of failure dividing the set of sites into a number of groups. The possible number of active groups characterizes the two classes.

A commit protocol is nonblocking if, in the event of any failure, all participating sites in the active groups (to be referred to as *active sites*) consistently complete the transactions which are incomplete at the time of failure. If an inactive site has aborted (committed), then none of the active sites commit (abort).

It is easy to notice that not all possible conditions that can be formed by the predicates and functions defined earlier are feasible under a given protocol. Assuming no failures, the execution of a commit protocol can be depicted as a sequence of conditions, first of them being the initial condition $(\forall i)(\text{Subtrans}(i))$. Any condition C' following a condition C in this sequence is derivable from C under the given protocol.

Formally, for any group S, let C(S,P) denote the set of all possible conditions satisfied by S under the protocol P. For disjoint groups $S_1, S_2, \ldots$, satisfying the conditions $C_1, C_2, \ldots$, respectively, we say that $\cup S_i$ is realizable under $\wedge C_i$ if and only if $C_1 \wedge C_2 \wedge \ldots \in C(\cup S_i, P)$. Let I be the set of all sites. A partition of I is a set of mutually disjoint groups $S_1, S_2, \ldots$ such that $\cup S_i = I$. A partition $\{S_1, S_2, \ldots\}$ in which S_i satisfies C_i is *realizable* under P if and only if $\cup S_i$ is realizable under $\wedge C_i$.

For a centralized protocol P, consider the group S that contains all sites except the central site. Since the sites are seperated in space, if there is no prefixed order in which the messages are sent to the sites, the following conditions are in C(S,P) :

Transaction(i)$\wedge\neg$Transaction(j),

(Decision(i)$\neq$'none'$\wedge$Decision(j)='none'), and,

(Complete(i)$\neq$'none'$\wedge$Complete(j)='none'), for all i,j$\in$S.

For a decentralized protocol P, the above conditions are in C(I,P). Thus, we have the following general property :

P5: for any commit protocol P, and for any group S not containing the central site, the conditions,

(Transaction(i)$\wedge\neg$Transaction(j)),

(Decision(i)$\neq$'none'$\wedge$Decision(j)='none'), and,

(Complete(i)$\neq$'none'$\wedge$Complete(j)='none') are in C(S,P), for all i,j$\in$S.

In the following, we specify the properties required by C(I,P) in order to have a nonblocking protocol P. The

following convention is used in expressing the formulas: for a bounded variable i, 'active i' refers to a site which is in an active group at that time, 'inactive i' refers to a site in an inactive group, while i can be any participating site, if unspecified.

Definition 3.3: Let P be a commit protocol. A *termination protocol* (TP) [SS 81] of P is any rule of action that transforms a realizable partition into a partition where $(\nexists i,j \in I)$(Complete(i)≠'none'∧Complete(j)≠'none' ∧Complete(i)≠Complete(j)) is true.

A termination protocol is *nonblocking* [2] if it transforms any realizable partition into a partition where (∀ active i,j)(Complete(i)=Complete(j)≠'none') is true.

A termination protocol which is not nonblocking is *blocking*.

If a commit protocol P has a nonblocking TP, then we say P is *nonblocking for failures* [SS 81].

The **nonblocking problem** for a class of failures is defined as : " Find a commit protocol nonblocking to the given class of failures." Our basic concern is to invesigate this problem and to relate the complexities of this problem for site failures and partitioning.

To start with, we prove an important negative result about the existence of nonblocking commit protocols. This result has been obtained earlier [SS 81].

Following is the general approach we take while proving the nonexistence of a nonblocking TP in a given situation : We

---

[2] Our nonblocking TP is the same as Skeen's TP. We generalize his notion to investigate the properties of TPs which are not nonblocking.

produce a group S satisfying a certain condition C and show that there are two other groups S',S" that could co-exist with S satisfying the conditions C',C". Then, we shall show that S' completes the transaction by committing while S" completes by aborting, under any TP f. Hence, it follows that S cannot complete the transaction, showing that there is no nonblocking TP.

Theorem 3.1. There is no commit protocol nonblocking to both the partitioning and site failures.

Proof: Let P be a commit protocol. By P5, $(\exists j,m)(\text{Complete}(m)=\text{'commit'})\wedge(\text{Complete}(j)=\text{'none'}))$. The partition $\{\{j\},I-\{j\}\}$ with this condition is realizable under P. Now, Decision(j) is either 'none' or 'commit' (by P3). Assume that it is 'none'. Then, no TP can make j 'abort' since it leads to inconsistency. But, Complete(j) cannot be 'commit' since Decision(j)='none' (by P2). Assume that Decision(j)='commit' and that i is not the supervisor. The partition $\{\{i\},\{j\},I-\{i,j\}\}$ is also realizable with the condition $(\text{Decision}(i)=\text{'none'}\wedge\text{Decision}(j)=\text{'commit'})$ (by P5). Since there can be no communication among different groups, a group has no way of knowing about the configuration of the other groups and hence the action of a TP can depend only on the sites in a group and the conditions satisfied. Now, Complete(i) cannot be 'commit'(by P2). Complete(j) cannot be 'abort' either (since Complete(m)='commit'). Thus, there is no nonblocking TP in this case also. □

In the remainder of this chapter, we answer the following questions:

a) when are commit protocols nonblocking to site failures?

b) are there simple commit protocols nonblocking to site failures with very small costs? (We will define the cost measures later)

c) can we make *some* of the groups proceed in the event of partitioning?

3.3. Optimal Nonblocking Commit Protocols

Let us examine one of the simplest protocols, P, with the assumption that site i has all the subtransactions [Gray 78].

Transaction-initiation rule:
Site i: Send the subtransaction to all participating sites j, where $j \neq i$.
Site j: Receive the subtransaction, process and commit it.

Reach-decision = Transaction-completion = Null.

It is not very difficult to see that P is a commit protocol. Commit protocols P', for which the condition $(\exists i,j \in I)(\neg \text{Transaction}(i) \wedge \text{Complete}(j)=\text{'commit'})$ is in C(I,P'), are known as *instant-commit* protocols. Observe that a participating site commits unilaterally under these protocols, as with P above.

However, P is a *blocking* protocol for site failures : Assume that i fails before it sends out all the subtransactions. Then the sites which have not received their subtransactions have to wait for the recovery of i. (Assume that such sites would be notified by one of the operational sites which has received the transaction.) In fact, this is true for any instant-commit

protocol.

Theorem 3.2. There is no instant-commit protocol nonblocking to site failures.

Proof: Consider the condition

$C = (\exists i,j \in I)(\neg Transaction(i) \wedge Complete(j)='commit')$ and the partition $\{\{i\}, I-\{i\}\}$ where $\{i\}$ is the only active group. By the hypothesis, this partition with the condition C is a realizable partition. But, Complete(i) cannot be 'commit' since $Complete(i)='commit' \rightarrow Transaction(i)$ (by P1). It cannot be 'abort' either since Complete(j)='commit'. □

Definition 3.4: A commit protocol P is *delayed-commit* if, the condition

$(\exists j,k)(Complete(j)='commit' \wedge \neg(Transaction(k))$

is *not* in C(I,P) .

Obviously, any commit protocol which is not instant-commit is delayed-commit.

Now, we define a measure for the cost of commit protocols which reflects the total time duration for the execution of a transaction under the given protocol.

Definition 3.5 : Let P be any protocol. Two messages in a transaction execution under P are *independent* if there is no causal dependency [Lamport 78] between them[3]
We define a *step* of P as a set of independent messages under P.
The *size* of a step is the number of messages in that set.
The *number of steps in an execution of a transaction T* under P is the minimum number of steps required for that execution of T.
The *number of steps in protocol P* is the maximum of the number of steps of all transaction-executions under P.

---

[3] In our context, a message M_1 is *causally dependent* on another message M_2 if and only if M_1 cannot be generated unless M_2 is received by the site originating M_1. A set of messages is independent if none of them is causally dependent on another, even transitively.

Note: Our notion of *step* clearly does not depend on the nature of the protocol. It is equally applicable for centralized, hierarchical, decentralized and any other kind of protocol.

It is now clear that instant-commit protocols typically are single-step protocols. Here, we have a simple lower bound on the cost of the delayed-commit protocols:

Theorem 3.3. Any delayed-commit protocol has at least two steps.

Proof: It follows from the definition of delayed-commit protocols that one step is needed to achieve (∀j)(Transaction(j)) (Transaction-initiation step) and one more step to achieve (∀j)(Complete(j)='commit') (Transaction-completion step). □

Observe that the following 'two-phase' protocol is a delayed-commit protocol :

> Transaction-initiation rule: Central site sends subtransactions of the transaction to each of the participating sites.
>
> Reach-decision rule: Null.
>
> Transaction-completion rule: After sending all the subtransactions, the central site sends the decision to each of the participating sites; each participating site completes accordingly.

From Theorem 3.3, we see that this protocol is optimal in the number of steps.

3.3.1 Delayed-commit Protocols under the centralized decision

We have introduced the centralized decision model in Section 2.5. Here, we express this notion formally under our model. We say that the Decision Making rule for a commit protocol P is

centralized if the following condition holds for all $C \epsilon C(I,P)$ under P:

P6. $(\forall k)(\text{Transaction}(k) \rightarrow \text{Decision}(k)=\text{'commit'})$.

The participating sites, after receiving the transaction, assume that the global decision is commit.

The centralized model of decision is very appropriate in the context of updates to replicated databases [Garcia 80] where the updates are normally computed at a single site and sent to the other sites which are not involved in the decision making process.

Theorem 3.4. Let P be a delayed-commit protocol in the centralized model of decision. Then, P is nonblocking to site failures.

Proof: For any realizable partition of I under P, $(\exists i \epsilon I)(\text{Complete}(i)=\text{'commit'}) \rightarrow (\forall j \epsilon I)(\text{Transaction}(j))$ (since P is delayed-commit). Define a TP, T as follows*:

Condition: $(\exists$ active $i)(\neg\text{Transaction}(i))$.
Action: Make all active i abort the transaction.
Justification: $(\exists i)(\neg\text{Transaction}(i)) \rightarrow (\nexists j)(\text{Complete}(j)=\text{'commit'})$.

Condition: $(\forall$ active $i)(\text{Transaction}(i))$.
Action: Make all active i commit the transaction.
Justification: $(\nexists$ j$)(\text{Complete}(j)=\text{'abort'})$ (by P3,P6). □

It is interesting that all delayed-commit protocols are nonblocking to site failures under the centralized decision. It

---

*Note: While defining a TP, we only present the actions and ignore 'how' they can be realized. Thus, the actual TP may have more number of steps than are indicated here since it needs to worry about further failures during its own execution.

is instructive to see how the participating sites can terminate a transaction in the absence of the central site : Their action critically depends on the assumption that a transaction can be aborted only when it is impossible to commit.

Let us consider the network partitioning now. By allowing some indirect communication among groups and restricting the number of possible groups, we can improve the situation for network partitioning as well. For example, when the partitioning involves exactly two groups and the undelivered messages are returned to the originating sites, nonblocking TPs are natural to expect [SS 81]. (Observe that this situation is similar to the no-failure situation.)

Normally, even though a protocol is blocking for partitioning, an active group with a site whose transaction has been completed can complete the transaction at all its sites. Similarly, if there is a site that has not received the transaction, all sites in that group can abort that transaction. Thus, we wish that there is at least one group such that (Complete(j)='none'^Transaction(j)) for all j in that group, which can complete the transaction after a partitioning. Ideally, one would like to see that all such groups can complete the transaction after partitioning. But, under the centralized decision model, this is equivalent to the nonblocking problem and hence has a negative answer.

Definition 3.6: Let P be a delayed-commit protocol. A TP that can transform *some* group S not containing the central

site[5] and satifying
$C = (\forall\, i \in S)(\text{Complete}(i)=\text{'none'} \wedge \text{Transaction}(i))$ into S
satisfying $C_0 = (\forall\, j \in S)(\text{Complete}(j) \neq \text{'none'})$,
is called a *non-trivial TP* for P.

We are interested in the problem, " Find a commit protocol which has a non-trivial TP," refered to as the *non-trivial termination problem* for partitioning. Since there are more than one potentially active groups, a group not having the central site cannot make any assumptions about the behaviour of the other groups. Thus, the answer to this query is negative :

Theorem 3.5. There is no delayed-commit protocol P with a non-trivial termination protocol under the centralized decision model.

Proof: Consider a group S satisfying
$C = (\forall i \in S)(\text{Complete}(i)=\text{'none'} \wedge \text{Transaction}(i))$. Consider two groups S', S" with the conditions
$C' = (\exists\, i \in S')(\text{Complete}(i)=\text{'commit'})$ and $C'' = (\exists\, k \in S'')(\neg \text{Transaction}(k))$. The partitions {S,S'} and {S, S"} are realizable with the conditions $C \wedge C'$ and $C \wedge C''$ respectively. Hence, there is no non-trivial TP for P. □

3.3.2 Delayed-commit protocols under Decentralized Decision

If it is possible that certain sites may not be able to complete their components of a transaction for whatever reasons, the transaction as a whole cannot be completed. Thus, we may be forced to abort the transaction and undo its effects on the

---

[5]This definition holds for both the centralized and decentralized protocols.

database to ensure the atomicity. In this case, we need to use the decentralized decision model. Here, the asymmetry between the commit and abort decisions is worth noting. Since the participating sites can possibly make individual local decisions, the global decision can be 'commit' only if *all* local decisions are 'commit'; in contrast, if *any* local decision is 'abort', the global decision *should* be 'abort' and the transaction can immediately be aborted at sites whose local decision is 'abort'. Formally, if P is a commit protocol such that the following condition is true for all $C \epsilon C(I,P)$, then we say that the decision is *decentralized*:

P7. $(\exists k)$(Decision(k)='commit')
$\rightarrow (\forall j)$(Transaction(j).

If a site makes an abort decision, then the transaction cannot be committed. Hence, such a site can unilaterally abort the transaction before notifying its decision. Thus, we have,

P8. $(\forall k)$((Decision(k)='abort') $\rightarrow$ (Complete(k)='abort')).

These conditions formally depict the asymmetry between the commit and abort decisions. Note that the receipt of the transaction component is not a prerequisite for a site to abort it. (In this case, there is no explicit abort of the transaction. By default, it is considered to be aborted.) Let us extend the two-phase protocol by introducing the Reach-decision rule as follows:

Reach-decision rule: Each participating site sends its local decision to the central site, central site finds the global decision. (Note that any participating site can abort the transaction immediately if its local decision is 'abort').

This is the centralized version of the Reach-decision rule. We can similarly give the decentralized version where each participating site sends its decision to all other participating sites so that each site can determine the global decision after receiving messages from all the other sites.

This three-step protocol is the same as the two-phase protocol in the centralized decision model, except that the participating sites make local decisions and send to the central site here. It has been shown [Garcia 79, SS 81] that it has no nonblocking TP. Let us consider that the central site and some of the participating sites fail simultaneously during the Transaction-completion action such that $(\forall \text{ active } k)(\text{Decision}(k)=\text{'none'} \wedge \text{Transaction}(k))$ holds in the active group. The transaction now cannot be completed at the active sites, because all the operational sites have made commit decisions locally and the global decision is not known to them.

Now, we present a set of necessary and sufficient conditions:

Theorem 3.6. Let P be a [illegible]yed-commit protocol under the decentralized decision mod[illegible]. Let the condition $(\exists\ i,j\epsilon I)(\text{Decision}(i)=\text{'none'}\wedge\text{Complete}(j)=\text{'commit'})$ be denoted by C. Then, there exists a nonblocking TP of P for site failures if and only if $C \not\epsilon C(I,P)$.

Proof: Assume $C\epsilon C(I,P)$. From P5, the condition $C_1 = (\exists\ i,j\epsilon I)(\text{Decision}(i)=\text{'none'}\wedge\text{Decision}(j)=\text{'abort})\epsilon C(I,P)$.

Consider the partition {{i},I-{i}} where {i} is the only active group. This partition is realizable under both C and C_1. Then, Complete(i) can neither be 'commit' nor 'abort'.

Conversely, assume $C \not\in C(I,P)$. Thus, for any realizable partition of I under P, by P8 and the fact that P is delayed-commit, we have

C_2 = ($\exists$ i)(Complete(i)='commit')$\rightarrow$($\forall$ j)(Decision(j)='commit').

Now, we can define a TP as follows:

Condition: ($\exists$ active i)(Decision(i)='commit').
Action: Make all active sites commit.
Justifiction: Property P7.

Condition: ($\nexists$ active i)(Decision(i)='commit').
Action: Make all active sites abort.
Justification: Condition C_2 above. □

These conditions are equivalent to Skeen's Nonblocking theorem [Skeen 81]. If it is possible that two sites i,j coexist such that i has committed and j has not received the global decision, j alone cannot decide whether the transaction can be committed or aborted. Any protocol under which such a condition is realizable is blocking. Now, looking back at the two-phase protocol under the decentralized decision model, it is easy to see that such i,j exist.

Given a commit protocol, we can consider the following simple procedure to determine whether it has a TP nonblocking to site failures or not :

a) Determine if the protocol is delayed-commit or not. If instant-commit, it is blocking.

b) Determine if the decision model is centralized or decentralized. If centralized, it is nonblocking.

c) Write the sequence of possible global conditions under the protocol. If one of them is C in the above theorem, then it is blocking.

d) Otherwise, it is nonblocking.

Since the site failures are quite frequent in any realistic environment, we consider the commit protocols nonblocking to site failures here and derive a lower bound on the number of steps.

Theorem 3.7. [Skeen 81] Let P be a delayed-commit protocol under the decentralized decision model. If P is nonblocking for site failures, then P requires at least 4 steps if it is centralized and 3 if decentralized.

Proof: Transaction-initiation needs at least one step. Since the decision is decentralized, reach-decision takes at least one more step. Let us assume that the decision is 'commit'. By the proof of Th. 3.6,
$(\exists k)(\text{Complete}(k)=\text{'commit'}) \rightarrow (\forall j)(\text{Decision}(j)\neq\text{'none'})$. Hence, it requires two more steps for the Transaction-completion in the centralized case and one in the decentralized. □

From Theorem 3.6, we see that P can be nonblocking to site failures only if the condition
$(\exists k)(\text{Complete}(k)=\text{'commit'}) \rightarrow (\forall j)(\text{Decision}(j)\neq\text{'none'})$ is true. Thus, it is not difficult to enhance any commit protocol not satisfying this condition to make it nonblocking to site

failures. We only need to modify the Transaction-completion rule as follows:

Transaction-completion rule:
Central site sends decision to all participating sites. If the decision is 'abort', the transaction is aborted at all the participating sites. If the decision is 'commit', the participating sites wait for another message, 'complete', after receiving which they commit.

This protocol is the same as Skeen's three phase protocol [Skeen 81] whose decentralized counterpart can be presented similarly. It is easy to check that this is nonblocking for site failures and takes 4 steps, becoming optimal under the decentralized decision model.

3.3.2.1 Fundamental relation among the failures

Under the centralized decision model, we have defined the notion of non-trivial TPs as a means of estimating the effectiveness of commit protocols in presence of partitioning. Here also, we would like to define a reasonable notion of non-trivial TPs. We wish to have *all* groups in which no site has the global decision to be able to consistently complete the transaction. This can be expressed as follows:

Definition 3.7: Let P be a (delayed-)commit protocol. A TP that can transform any group S not containing the central site and satisfying
C = $(\forall i \in S)$(Decision(i)='none'$\wedge$Transaction(i)) into S satisfying the condition C_0 = $(\forall i \in S)$(Complete(i)$\neq$'none') is called a *non-trivial TP* of P.

Notice that this condition is not equivalent to the nonblocking requirement. Specifically, we have left out the groups S satisfying the condition
$(\exists i \in S)(\text{Decision}(i)=\text{'commit'}) \wedge (\forall j \in S)(\text{Complete}(j)=\text{'none'})$.

Now, we derive the conditions under which a commit protocol in this model has a non-trivial TP. Recall that, in case of site failures, the hardest condition a TP could face was $(\forall i \in S)(\text{Decision}(i)=\text{'none'} \wedge \text{Transaction}(i))$, where S is the only active group. It can be nonblocking only if it could complete the transaction for all such S. Irrespective of whether any of the failed sites have received the global decision or not, a TP can complete in such cases if it is guaranteed that no failed site has already committed. For partitioning, we require that no site outside the given group is allowed to commit, if there is to be a non-trivial TP. This is equivalent to the condition that, a) no site outside the given group has already committed, and, b) no site is allowed to commit now. The second part of the condition can be guaranteed by a TP, leaving the first part to be handled by the commit protocol. Thus, the following result should not be any surprise :

Theorem 3.8. Let P be a (delayed-)commit protocol with the decentralized decision. Then, P has a non-trivial TP if and only if the condition
$C = (\exists\ i,j \in I)(\text{Decision}(i)=\text{'none'} \wedge \text{Complete}(j)=\text{'commit'}) \notin C(I,P)$.

Proof:

Necessity: Assume C is in C(I,P). As in the proof of Theorem 3.6,

we can easily produce a partitioning in which the transaction cannot consistently be completed.

Sufficiency : Assume $C \notin C(I,P)$. Now, we define a set of actions as follows:

Condition: $(\exists i \in S)$(Decision(i)='none') $\wedge (\forall j \in S)$(Transaction(j)).
Action: Make all sites in S abort.
Justification: Since $C \notin C(I,P)$, $(\nexists j \in I)$(Complete(j)='commit') at the time of partitioning. We will see below that there can be no such j even after the partitioning.

Condition: $(\forall i \in S)$(Decision(i)='commit' $\wedge$ Complete(i)='none').
Action: Wait until a reconfiguration.
Justification: We cannot abort for trivial reasons. Nor can we commit since it would conflict with the above abort decision.

These two actions guarantee that there are no simultaneous conflicting decisions, showing that it is a TP. The first action proves that it is non-trivial. □

Though the proof of this theorem is not involved, the significance of this result is far reaching. Firstly, observe that any protocol nonblocking to site failures has a non-trivial termination as well. (This is because the conditions in Theorems 3.6 and 3.8 are identical.) Thus, it relates the site failures to partitioning in a very curious and in a rather strong sense. We can restate this thorem as the following basic result:

Fundamental theorem on failures: The nonblocking problem of site failures is equivalent to the non-trivial termination problem of partitioning.

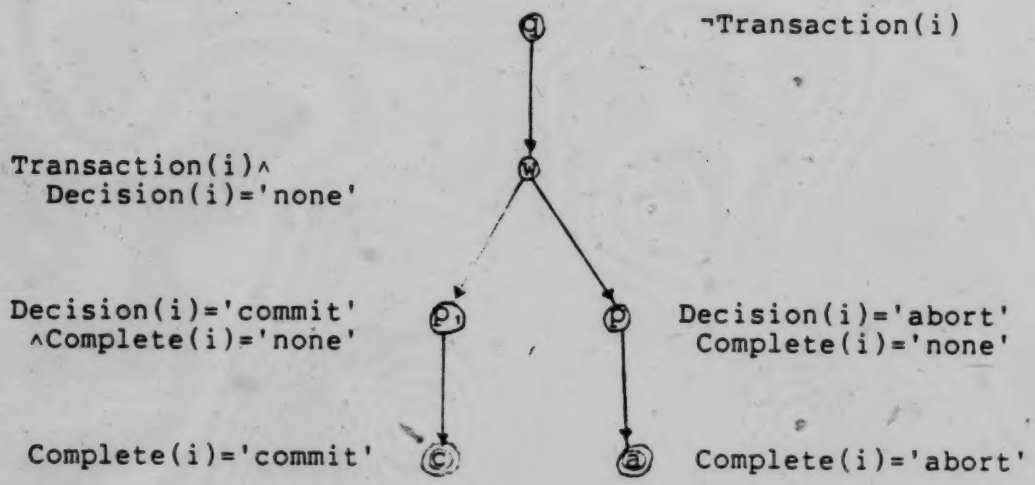
Thus, any solution to the nonblocking problem of site failures is a solution to the non-trivial termination problem of partitioning and vice versa. It is a theoritical proof that the problem of site failures is properly embedded in the problem of partitioning in an explicit form, showing that partitioning is intrinsically much harder than site failures. Our basic contribution lies in demonstrating the subproblem of the partitioning the site failures are equivalent to. Interestingly, our notion of non-trivial TPs brings out all the common aspects of these two problems.

Another useful consequence of this result is that any commit protocol satisfying the conditions of Theorem 3.8 can be picked up to study the non-trivial TPs in the presence of partitioning. Several consequences of this result are explored in Chapters 6 and 7.

3.4. Correspondence with FSA model

The basic structure of our model for commit protocols corresponds to the basic structure of Skeen's FSA (finite state automata) model: the conditions satisfied by individual sites in our model can be matched with the states of FSA and vice versa. The condition (¬Transaction(i)) corresponds to an initial state q, the condition (Transaction(i)∧Decision(i)='none') corresponds

to a state w adjacent to q and the condition (Complete(i)≠'none') corresponds to the final states c('commit') and a('abort') depending on the value of Complete(i). Under the Decentralized Decision model, the condition (Decision(i)='commit'∧Complete(i)='none') corresponds to a new state p_1 adjacent to w and c while the condition (Decision(i)='abort'∧Complete(i)='none') corresponds to a state p_2. In a similar fashion, we can assign conditions to each state in an FSA.



3.5. Read-only Transactions

Here we try to take advantage of special properties of transactions to improve the performance of the protocols. A large fraction of the transactions that access the databases tend to be database queries. Thus, these transactions are *read-only* in their nature.

A read-only transaction expects some information to be output to the user and hence, can be completed only if the originating site receives all the required information from the participating sites. As soon as a participating site has sent its share of the information, the participating site can consider that transaction as 'commit'ted' since it is not affected even if the global decision is abort. Thus, in case of read-only transactions, *sites can unilaterally commit* the transactions and the global decision is commit only if all the participating sites commit. Notice that the atomicity is maintained in this case even if some sites commit while others abort a transaction. Now, we present a simple, nonblocking protocol to deal with the read-only transactions:

> step 1: Central site sends out the participation requests. Sites receiving the requests process them, commit or abort unilaterally, depending on the outcome of the processing.
> step 2: A participating site sends back the result of the processing if it has committed. Otherwise, it sends a 'no'. The central site, after receiving the responses, commits if all of them have committed and aborts otherwise.

Correspondingly, we have a simple TP as follows:

> If the central site has received the information from all sites, it commits. Otherwise, it aborts.

Observe that the above TP can deal with the partitioning also : participating sites unilaterally abort or commit while the

central site commits only if it has received the information from all the participating sites.

This is no contradiction to our previous results - the protocols we have given here are not 'commit' in the sense of our definition of commit protocols. But, importantly, these protocols also guarantee the consistency of the databases.

Till now, we have considered only the central control transaction model. Extension of the above arguments to the hierarchical and nested models is not straight forward. In the hierarchical model, a (sub)transaction is aware of only its immediate predecessor and immediate successors. We know that an update cannot be permanently incorporated into the database unless the whole hierarchy commits the transaction. Hence, a sub-transaction cannot be unilaterally committed if one of its descendants is not read-only. But, there is no way of verifying this fact at the time of transaction-initiation. Thus, the problem is in determining whether a given sub-hierarchy is read-only or not. If there is such a mechanism available to the root of the sub-hierarchy, then the root can behave exactly as the central site in the above protocol.

A simple scheme is as follows: Leaves in the hierarchy determine whether their subtransaction is read-only or not and notify to their parents. A parent who receives 'read-only' notifications from all its immediate successors thus knows that its sub-hierarchy is read-only.

The complete protocol to be followed by a parent and its descendents can now be given :

step 1: The parent sends participation requests. The children determine whether their subtransactions are read-only or not.

step 2: A child sends back its share of the information and a 'read-only' message if its sub-hierarchy is read-only. Otherwise, it sends its decision on the transaction. The parent immediately commits if it has received 'read-only' messages from all its children.

Thus, in read-only sub-hierarchies, we have allowed unilateral aborts and unilateral commit. This feature obviously reduces the number of messages in the protocol. Specifically, at least one set of messages, carrying the global decision down the hierarchy, are completely eliminated in the read-only sub-hierarchies.

Site failures can also be easily handled : if a site fails before it has notified whether it has a read-only sub-hierarchy or not, then the whole transaction is aborted. If a site fails after notifying that it has read-only sub-hierarchy, then its failure is ignored.

The same solutions work for the nested transaction model also since the essential difference between hierarchical and nested transactions is the synchronization among the subtransactions.

3.6. Summary

The results obtained in this chapter can be summarized as follows :

Commit protocol	Nonblocking to site failures	Has non-trivial termination pr.	No. of steps
Instant-commit	No	No	1
Delayed-commit, Centralized dec	Yes	No	2
Decentralized decision,<4 stp	No	No	3
Satisfying cond. of Th.3.6	Yes	Yes	4

Chapter 4 : Site failures and Recovery

Contents

4.1. Introduction

4.2. Bounded failure Networks,

4.2.1 Nonblocking protocols

4.3. Failure recovery

4.1. Introduction

In the previous chapter, we have introduced a new information-based model for commit protocols and derived several existence results on the nonblocking aspects. We have implicitly assumed that a failure can affect any of the sites and any number of the sites can be affected at the same time. In this chapter, we study the site failure problem, restricting the assumption of arbitrary number of simultaneous failures. We do not consider restricting the simultaneous failure of specific sites (of the type, 'site i cannot fail if site j has already failed') since it is very similar to the restriction on the number and the results similar to what we obtain here are trivially derivable in that case also.

We strongly feel that the assumption of arbitrary number of site failures is *not* realistic, particularly with the growing

sizes of networks and the growing reliability of individual nodes. For instance, if there are 10 sites participating in a transaction, we do not expect 9 of them to fail simultaneously. Here, we propose a bounded-failure network model which, we believe, is closer to reality. We will see that this model not only reduces the number of messages in optimal nonblocking commit protocols, but also induces definite advantages in dealing with the transaction-completion and site recovery.

We then study the recovery aspects of sites. We present the possible recovery strategies sites can use and characterize the commit protocols into classes that allow these strategies. We shall present several problems that arise in the context of these recovery strategies.

4.2. Nonblocking Protocols for Bounded-failure Networks

Definition 4.1[Lamport 78]: Two sites are said to have failed *simultaneously* if each of them fails before knowing about the failure of the other.

The extention of this definition to k sites is straight-forward and the *k-bounded-failure model of networks* has the following properties:

1) At most k sites fail simultaneously, for $k \ll n$, and,
2) If there are $m > k$ failed sites at some time t, then there exists $p \leq k$ such that there was a duration for performing one step between the failure of p th and p+1 th sites.

We would like to clarify our bounded-failure concept vis-a-vis k-resiliency introduced by Alsberg et al [AD 76]. K-resiliency is the property of a commit protocol while bounded-failure is a property of the network. If a commit protocol is k-resilient, it ensures that a nonblocking TP exists as long as not more than k sites fail during a critical stage of the transaction-execution. In contrast, if a network is of k-bounded-failure, no more than k sites can fail simultaneously; however, any number of them can fail at any stage of the transaction-execution.

4.2.1 Nonblocking protocols

Under the bounded-failure model, one can always take advantage of the fact that at most k sites can simultaneously fail to ensure that, for any $C_o \epsilon C(P)$, C_o contains sufficient information for P to be nonblocking. A scheme we can use for this purpose is similar to the SDD-1 scheme[HS 80]: Designate k sites as first-level backups to the coordinator. (For an elegant implementation of backups, see [ABG 81].) The coordinator sends the necessary information to its k backups before communicating with any of the participating sites and after any new information is obtained. If there are m failures among the k+1 sites at some point of time, the coordinator (if operational) or one of the backups sends the necessary information as known to it to a set of m second level backups and the list of these new backups to the coordinator (if it is not down). Since $k<<n$, this can also be carried out in parallel with the coordinator's actions and the

transaction-execution time would not be increased by any appreciable amount [HS 80].

Throughout this section, we assume P to be a centralized commit protocol. Observe that the bounded-failure model is not applicable in the context of the decentralized protocols.

It is easy to see that, if the central site can send all the subtransactions at the beginning of the transaction-execution to each of its backups, the bounded-failure model can give us simple commit protocols nonblocking to site-failures.

Theorem 4.1. There is a two step commit protocol nonblocking to site failures under the k-bounded-failure network model, such that the size of one of the steps is O(k).

Proof: With backups, we can ensure that all the subtransactions are contained among the active sites at any point of time, ie., $(\forall C_o \epsilon C(P))(C_o \rightarrow$ (∀ active i)(Subtrans(i))). Thus, in effect, the condition
(∃j, active i)(Complete(j)='commit'∧¬Transaction(i)) is not realizable. This implies that the instant-commit protocol of Section 3.3, with this modification, satisfies the property of a delayed-commit protocol under centralized decision. By Theorem 3.4, it is nonblocking to site failures. □

We should note that this protocol is very efficient, particularly in the context of replicated databases. We consider it to be much cheaper to send all the subtransactions to the backups (which are usually very few in number and need not be

different from the participating sites) than to have a two-step commit protocol.

We have the following interesting result concerning the delayed-commit protocols:

Theorem 4.2. [HS 80] Under the k-bounded-failure network model, any delayed-commit protocol with decentralized decision can be modified so that it is nonblocking to site failures with the introduction of one extra step of size O(k).

Proof: 'Decision' is the information that the coordinator would send to its backups. Initially, the backups assume 'abort' as the decision by default and when the coordinator finds that the transaction can be committed, they are informed of the change. By P4 of Chapter 3, if there is an active site with the decision, it can be passed on to the other active sites. Thus, the conditions of Theorem 3.6 are easily satisfied, leading to a nonblocking TP. The number of extra messages introduced is k or 2k (depending on the decision) if any one of the k+1 sites survive throughout the transaction-execution. □

The above results show that inexpensive commit protocols nonblocking for site failures can be designed under the bounded-failure model.

The application of this model leads to inexpensive Termination Protocols as well. First, let us consider the two-phase protocol under the decentralized decision. We designate k backups to the central site in a predefined order.

The central site sends the transaction components and the list of backup sites to the participating sites. When it decides to commit the transaction, the decision is first sent to all the backups and then to the participating sites. If the failure of some sites is detected, the central site (assuming to be operational) designates an equal number of new backups, passes on the decision if it is made. If the central site also fails, the first backup to be operational performs this task. Thus, if there are failures at any point of time, (k+1) sites are first ensured to have the decision if it is made. If no decision is made, it is taken as abort.

The arbitrary failure model has a worst case bound of $O(n^2)$ on the number of *phases* for a decentralized TP and O(n) for the centralized [Skeen 81a]. (Recall that a phase is almost equal to two steps.) Here, we see that atmost O(n) *messages* are sufficient in a nonblocking TP for any number of site failures.

Another benefit of our model is as follows: One always prefers that a transaction can never be aborted if one of the sites (even a failed site) knows of the global decision to be commit. In the TPs in Section 3.3.2, we have aborted the transaction if none of the operational participating sites know the global decision to be commit, though some failed sites do. Under the bounded model, the backups are guaranteed to know the global decision to be commit in such a case and hence can direct to commit the transaction. Thus, more transactions can possibly be committed under this model than in the arbitrary failure

model.

It has been shown by Alsberg et al [AD 76] that the probability that more than two sites simultaneously fail is negligible in most practical environments. Thus, by choosing proper k, one can reduce this probability to arbitrary small levels. In fact, the central site, depending on the number of participating sites and any statistical information available on the failure of these sites, can compute the value of k for each transaction at the time of its initiation, ensuring the probability that more than k sites fail simultaneously is negligibly small. Notice that our scheme fails only if a) *all* the back-ups and the central site fail simultaneously, b) the decision model is decentralized, and, c) none of the active participating sites know the global decision. One would expect such a situation to be non-existent for all practical purposes. Hence, when such an occassion arises, we can let the sites wait until the recovery of enough sites to proceed.

4.3. Failure Recovery

In our formalism, failure recovery is equivalent to the merging of different groups. If two groups being merged are both active, two possiblities arise - a) one of the groups has completed the transaction, or, b) both groups are waiting. In the first case, the group that has not completed (if any) communicates with sites in the other group and completes the transaction (by P4 of Chapter 3). The latter case is only an instance of partitioning and thus, requires the execution of a TP

for this new group. Hence, we do not discuss the recovery aspects of partitioning here. On the other hand, recovery in the context of site failures is interesting in itself.

Assuming that sites do not lose information that they possessed before failure, when a failed site recovers, it contains information regarding the status of transactions it was participating in, at the time of its failure. Depending on the commit protocol and the TP used, the recovering site can deduce some information regarding the other sites' behaviour. According to the information a site has to acquire from an external source in order to complete the transaction, we can classify the commit protocols as follows:

Let P(i) denote the conditions satisfied by any failed site i at the time of its failure.

Definition : A commit protocol is *absolutely recoverable* iff, for any i and P(i), one of the following is true:
i) $P(i) \rightarrow (\exists\ k)(\text{Complete}(k)=\text{'commit'})$,
ii) $P(i) \rightarrow (\exists\ k)(\text{Complete}(k)=\text{'abort'})$.

It is *relatively recoverable* iff it is not absolutely recoverable, and, if $(\exists\ k)(\text{Complete}(k)=\text{'commit'})$, then $P(i) \rightarrow \text{Transaction}(i)$.

It is *restrictively recoverable* iff it is not relatively recoverable.

Informally, a commit protocol is absolutely recoverable if a failed site can recover without communicating with any other site. It is relatively recoverable if a failed site can recover after communicating with *any* other participating site to find the decision taken by the active sites. It is restrictively recoverable if the failed site can recover only after

communicating with a specific set of sites.

The *independent recovery* [SS 81], where a site can recover without communicating with any other site, is simply the best possible strategy for recovery. Following is a characterization relating the commit, termination and independent recovery aspects of a protocol:

Theorem 4.3. A site i can recover independently under a commit protocol P and a TP f of P if and only if f *cannot* successfully terminate whenever P(i)=(Complete(i)='none').

Proof: Sufficiency is trivial since i can abort whenever Complete(i)='none'. If f successfully terminates in absence of i, and (Decision(i)='commit'), then i has no way of infering the mode of termination. (Note that i does not know which sites were operational when the transaction was terminated.) □

This result provides a natural trade-off between termination and recovery. A simple but profound consequence of this theorem is that no commit protocol nonblocking to site failures allows any site to recover independently. In other words, a nonblocking commit protocol can either be employed in conjunction with a nonblocking TP and eliminate the possibility of independent recovery for any site or a blocking TP is used allowing possible independent recovery. But, observe that the central site only (if it exists) can possibly recover independently. Hence, the following negative result is of no surprise :

Theorem 4.4. [SS 81] There is no absolutely recoverable commit

protocol.

Proof: Recall that an absolutely recoverable protocol allows *all* sites to recover independently. Hence, a commit protocol is absolutely recoverable if and only if it is nonblocking to partitioning. □

Relative recovery is a realistic strategy since we can expect to have at least one of the other participating sites to be operational when a site recovers from failure. A simple observation on this strategy is that a recovering site can expect to get only the global decision from other sites. Intuitively, this means that this strategy is not effective under the centralized decision model. The result to follow formalizes these observations.

Theorem 4.5. Let P be a delayed-commit protocol under the *arbitrary failure* model. If decision making is decentralized, then P is relatively recoverable and otherwise, it is restrictively recoverable.

Proof: P can be restrictively recoverable only if $(\exists\ k)(\text{Complete}(k)=\text{'commit'})\wedge(\exists\ \text{failed}\ i)(\neg\text{Transaction}(i))$ is true (after the execution of the termination protocol). However, if the decision making is decentralized, $(\exists\ k)(\text{Complete}(k)=\text{'commit'}) \rightarrow (\forall\ j)(\text{Transaction}(j))$ (by P7 of Chapter 3), contradicting the above condition. □

Intrinsically, the relative recovery requires that *all* operational sites should 'remember' the decision on any

transaction they have participated, until they are sure that all other participating sites have received the decision. Since this period can be indefinite, the sites need to retain the histories of past transactions long after they are completed. It is an open qustion to see if an inexpensive strategy can be worked out to deal with this problem.

From the above theorem, it follows that the restictive recovery is to be used only under the centralized decision model. In this case, if there are a number of designated sites, any recovering site has to communicate with all of them and has to wait if any of them is not operational. This problem cannot be eliminated even if we insist that a new transaction requiring a failed site cannot be initiated and/or that the network returns messages to the originator if the destination site is down.

The bounded-failure model can solve this problem in a straight-forward fashion :

Theorem 4.6. Let P be a delayed-commit protocol under the *bounded-failure* model. Then P is relatively recoverable.

Proof: It is trivial when the decision is decentralized (Theorem 4.5). Assume the decision is centralized. In a bounded-failure network,
$(\exists$ active k)(Decision(k)$\neq$'none') holds throughout the transaction execution. Hence, we can always use a TP such that, if
$(\exists$ failed i)($\neg$Transaction(i)), then
$(\forall$ active k)(Complete(k)='abort'). Thus, P cannot be

restrictively recoverable. □

These results show that, any commit protocol under the centralized decision model can be made effective in its recovery aspects by slightly increasing its cost. But, protocols in the decentralized decision model do not gain anything in their recovery strategies. Thus, whether the commit protocol used is the Three-phase commit or the Two-phase commit with back-ups, the recovery strategies are identical : any failed site can recover by communicating with any one of the operational participating sites.

Chapter 5 : Network Partitioning - Preliminaries

Contents

5.1. Introduction

5.2. Problem Specification

5.3. Two approaches to the problem

5.4. The structure of FSA

5.5. Termination protocols

5.5.1 Commit protocols with several TPs

5.1. Introduction

In a distributed system, nodes communicate via the communication network. Typically, each node can communicate with all other nodes since there are paths in the network from a node to all other nodes. A message originated at a node i and destined to a node j normally goes through a number of other nodes on a path between i and j before it reaches j. If sufficiently many nodes or/and links fail, a subset of the nodes may not be able to communicate with the other nodes. This situation is known as *network partitioning*. We have shown in Chapter 3 that there is no commit protocol nonblocking to arbitrary partitioning. Thus, given a commit protocol, there exists a partitioning in which at least one group has to wait on an incomplete transaction until some repairs are made. Later, we will prove even stronger result - for any commit protocol, there

is a partitioning in which *all* except possibly one, groups wait. This shows that the worst case behaviour of all commit protocols is equally undesirable. In Chapters 6 and 7, we concentrate on the expected case behaviours and show that we can do considerably better.

5.2. Problem Specification

In the context of network partitioning, we ask the following questions:

a) Can we make *some* groups complete the transaction in *all* cases? (We answer this in the negative)

b) What is the best we can do in the average case?

We know that instant-commit protocols and delayed-commit protocols in the centralized decision model are incapable of dealing with the partitioning. We consider only the delayed-commit protocols in the decentralized decision model. We refer to them simply as 'commit protocols' in this and the next chapters.

5.3. Two approaches to the problem

Assuming that two groups cannot make inconsistent decisions, there are two possible approaches to solve our problems - a) assuming that there is a mechanism for detecting the partitioning, design techniques of handling incomplete transactions, and, b) design techniques to work irrespective of whether a partitioning is detected or not. The first approach envisages the following ordered sequence of events - a) a commit

protocol is being followed by the sites participating in a transaction, b) some components fail and all unaffected sites are notified, c) all notified sites switch on to a termination protocol, and, d) affected sites switch on to a recovery protocol after recovery from failure. Under the second approach, no failures are detected and no termination protocol is required. The regular commit protocol is followed by all sites unaffected by the failure and the recovering sites use a recovery protocol.

As an example of the second approach, let us consider the quorum-based protocols proposed by Skeen [Skeen 82a]. Quorum is a generalization of the majority-consensus: two values a and c are determined either by the system or the system administrator such that their sum exceeds the total number of sites where the transaction is executed. To start with, if a site decides to abort, the transaction is aborted. If all sites decide to commit, then they move into a 'prepare to commit' state. Each participating site now votes for 'commit' or 'abort' according to the rule that it votes for 'commit' only if it is in the 'prepare to commit' state. These votes are collected (either in a centralized, hierarchical or a decentralized fashion) and a 'commit' decision is made if the votes for 'commit' are no fewer than c. An 'abort' decision is made if there is no commit quorum and the 'abort' votes are no fewer than a. If neither decision can be made, the sites have to wait. The key to the correctness of this scheme is the requirement that $a+c > n$ where n is the number of participating sites. This makes sure that both 'commit' and 'abort' decisions can never be taken simultaneously

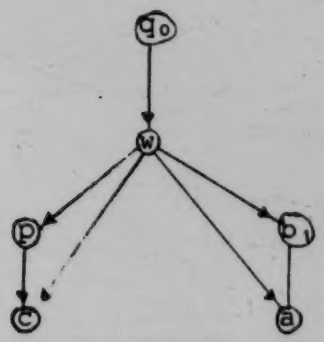
by different groups, though it is possible that more than one group can complete the transaction (making the same decision).

In this scheme, a decision can be made as long as the votes for one of 'commit' and 'abort' exceed c or a respectively. This can happen in spite of a number of site failures or even partitioning, the major advantage of such a scheme. Probably the most important disadvantage of these protocols is that they cannot take any advantage of failures less serious than partitioning. As we have seen in Chapter 3, site failures are less complex and more frequent than partitioning and there are commit protocols nonblocking to site failures.

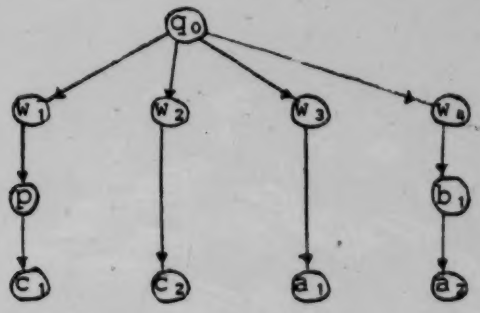
In the rest of our work, we shall take the first approach and assume that there are some means of detecting the classes of failures we are interested in. The model we use for commit protocols is the FSA model described earlier. We denote the initial states of an FSA by Q_0, the set of states that lead only to states in Ab by Pa, the set of states that lead only to states in Com by Pc, and the set of those that lead to both by N.

5.4. The structure of FSA

The FSA representing commit protocols have traditionally been very restricted in their structure. In fact, all the protocols that have appeared in the literature can be trivially modified, if necessary, to make their FSA tree-structured. For example, let us consider a decentralized protocol represented by the following FSA:



The following tree-structured FSA is equivalent to it in the conventional sense that they both accept the same language:



Without loss of generality, we focus our attention on tree-FSA.

Since we deal with centralized and decentralized protocols only, the FSA corresponding to these protocols are fairly uniform - they are all identical for all sites in a decentralized protocol and for all participating sites in a centralized protocol. We consider the states in identical FSA to be the same and say that the FSA in a protocol are *synchronized within a state* if a state can be concurrent only with the states adjacent to it in the state diagrams. Thus, if a site has reached a final state, any other site is either in a final state or in a state adjacent to a final state.

Theorem 5.1. Let P be a commit protocol represented by tree-FSA and synchronized within a state. Then, all its FSA are isomorphic.

Proof: Let F_1, F_2 be two FSA of P. For any path from the initial state to a final state of F_1, there corresponds a path in F_2 from its initial state to a final state, of the same length, since they are synchronous within a state and both are trees. (For the centralized protocols, such path lengths may differ by one. However, they can be made equal by introducing a dummy state, if necessary.) Furthermore, as there is a unique input associated with each transition in any FSA, for any two different such paths in F_1, there correspond two different paths in F_2. Hence F_1 and F_2 are isomorphic. □

Without loss of generality, we assume that all FSA have the *same* states.

5.5. Termination Protocols

Given a protocol synchronized within a state, any two sites operating under that protocol, and that one of them is in a specific state, then the other site can only be in one of a few states in the FSA concurrently.

Definition 5.1. Let P be a protocol. The **concurrency set** of state s at site i, denoted by R(i,s), is the set of ordered pairs (j,t) where site j can potentially be in state t when site i is in state s.

Since we are currently interested in the partitioning problem, we want to formally model a component (a group together with the status of the sites in it) , capturing all the information available in a component. Intuitively, we can see two kinds of information - the set of sites in the component and the status of the transaction at each of the sites (represented by the states). Since there is no possible communication with any site outside the component, no other information is available.

Let I be the set of sites in the network and Q be the set of states in the FSA. Let J denote the powerset of IxQ. Let D be the largest subset of J such that a component $C \epsilon D$ iff
i) $(i,s) \epsilon C$ & $(i,t) \epsilon C$ implies s=t, and
ii) $(i,s),(j,t) \epsilon C$ implies $(j,t) \epsilon R(i,s)$ & $(i,s) \epsilon R(j,t)$. (That is, s,t should be adjacent in the FSA)

Any set C satisfying these conditions represents a physically realizable component in a partition. Furthermore, all the information available in the physical component has been extracted into the formalism. Thus, we call D the set of all *realizable* components in the system. For any C in D, $site(C)=\{i \mid (i,s) \epsilon C\}$ and, $state(C)=\{s \mid (i,s) \epsilon C\}$. Extending the above definition of concurrency, two components C,C' can be *concurrent* only if $site(C) \cap site(C')=\emptyset$ and the states have concurrency relations among themselves. We formalize this concept of concurrency by requiring that $site(C) \cap site(C')=\emptyset$ and $C \cup C' \epsilon D$.

A *Termination protocol* (TP), can be viewed as a mechanism whose inputs are components and outputs are the decisions to be followed by those components. It has to ensure that no two components that could potentially occur concurrently in a partition are given conflicting decisions. Formally, we see a TP as a function whose domain is the set of all physially realizable components and co-domain is the set of possible decisions.

In addition to the two modes of transaction completion, commit and abort, we have one more possibility here, 'waiting'. Let x,y,w respectively represent these three possible actions, commit, abort and wait. Thus, a TP is a function from D to {x,y,w}. But, not all such functions can be termination protocols.

The primary requirement for a termination protocol is that it consistently terminates a transaction when failures occur. Observe that a component can be mapped to x or y as long as one of its sites is in a state from ($Com \cup Ab \cup Pa \cup Q_o$). In fact, the only states that require attention are those in Pc adjacent to states in N and those in N adjacent to states in $Com \cup Pc$.

Definition 5.2: Let P be a commit protocol. Let $Q = Q_o \cup \{\alpha \in N \mid \alpha$ is not adjacent to a state in $Com \cup Pc\}$ and $Q' = \{\beta \in Pc \mid \beta$ is not adjacent to a state in N$\}$. Let $f: D \rightarrow \{x,y,w\}$ be any function. We say f has *preservation property* iff for any $C \in D$,
i) $state(C) \cap (Com \cup Q') \neq \emptyset \rightarrow f(C)=x$,
ii) $state(C) \cap (Ab \cup Pa \cup Q) \neq \emptyset \rightarrow f(C)=y$.

f has *consistency property* iff for any two concurrent components $C,C' \in D$ (ie., $site(C) \cap site(C') = \emptyset$ & $C \cup C' \in D$), $\{f(C),f(C')\} \neq \{x,y\}$

f is a *termination protocol* (TP) of P iff f has both preservation and commit properties.

5.5.1 Commit protocols with several TPs: Here, we focus our attention on non-trivial termination protocols characterized by the following definition.

Definition 5.3: Let P be a commit protocol, f be a TP of P and n be the number of sites. We say f is *non-trivial* iff $f(C) \neq w$ for all $C \in D$ such that
$|C| \leq n-1$, central site$\not\in$site(C)*, and, state(C)$\subseteq$\{s|s$\in$N and is adjacent to a state in Pc$\cup$Com\}.

Following theorem is the counterpart of the Theorem 3.8, which establishes the relation between the nonblocking problem of site failures and the non-trivial termination problem of partitioning. Here, we present it in terms of the states of FSA.

Theorem 5.2. Let P be a commit protocol. Then P has a non-trivial TP if and only if no state in N is adjacent to a state in Com.

Proof:
Necessity: Assume that a state in N is adjacent to a state in Com and there is a non-trivial TP f of P. Consider C such that state(C)={s,t} where s is adjacent to a state in Com and t is adjacent to a state in Ab$\cup$Pa. Assume, without loss of generality, that f(C)=x. Since $|C| \leq n-1$, there exists a site i such that $(i,s) \in C$ and a site
k$\not\in$site(C) such that (i,s) is concurrent with (k,a) for some $a \in$Ab$\cup$Pa. But, since f is a TP, f({(k,a)})=y, showing that

---

*This definition is equally applicable for centralized and decentralized protocols.

$f(C)\neq x$. Similarly, we can prove that $f(C)\neq y$.

Sufficiency: Assume that no state in N is adjacent to a state in Com. Then, we define $f:D \rightarrow \{x,y,w\}$ as follows:

For $C\epsilon D$, $f(C)=x$ if $state(C)\cap Com\neq\emptyset$,

$f(C)=y$ if $state(C)\cap(Ab\cup Pa\cup N\cup Q_o)\neq\emptyset$, and,

$f(C)=w$ otherwise.

Clearly, f has the preservation property.

Assume that $\exists C,C'\epsilon D$ such that $(f(C),f(C'))=(x,y)$. Since no member of Com is adjacent to any member of $(Ab\cup Pa\cup N\cup Q_o)$, $C\cup C'\not\in D$. Hence, f is a TP of P. f is non-trivial since, $f(C)=y$ whenever $state(C)\cap N\neq\emptyset$. □

Thus, if $Pc=\emptyset$, then a commit protocol has no non-trivial TP. Below, we see that the condition $Pc=\emptyset$ has even stronger implications. Intuitively, if $f(C)\neq w$ for any f and $C\epsilon D$ such that central site$\not\in$site(C) and $state(C)\subseteq\{s\epsilon N \mid s$ is adjacent to a state in Com$\}$, then, $f(C')\neq w$ for any C' with similar conditions. Hence, either $f(C)\neq w$ for all such C or none of them. If $f(C)=w$ for all such C, then f satisfies exactly the requirements in the definition of a TP, thus showing that there is only one TP. The following result shows that this argument holds both ways :

Theorem 5.3. Let P be a commit protocol. Then, P has only one TP if and only if $Pc=\emptyset$.

Proof:

Necessity: Assume that $Pc\neq\emptyset$. Then, we can define $f':D \rightarrow \{x,y,w\}$ as follows:

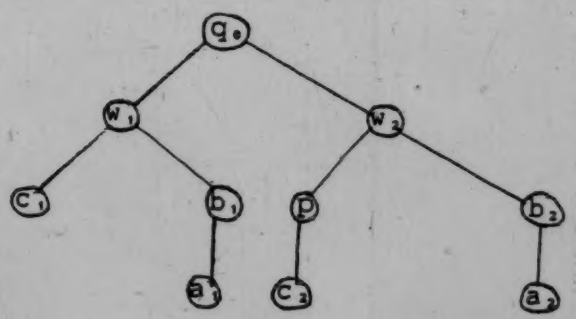
For $C\epsilon D$, $f'(C)=x$ if $state(C)\cap(Com\cup Pc)\neq\emptyset$,

f'(C)=y if state(C)∩(Ab∪Pa∪Q_0)≠∅, and,

f'(C)=w otherwise.

As in the proof of Theorem 5.2, f'≠f can be shown to be a TP. Sufficiency: Assume that Pc=∅. Then, we can easily show that any TP f" coincides with f. □

Let us note the difference between Theorems 5.2 and 5.3 : if no state in N is adjacent to a state in Com, then Pc≠∅. But, the converse is false. For instance, let us consider the decentralized commit protocol defined by the following FSA :



Here, Pc={p}. But, the state w_1 is adjacent to c_1∈Com as well. For this commit protocol, it is straight forward to define a second TP.

It follows that the class of protocols satisfying the conditions of Theorem 5.2 is a proper subset of the class of commit protocols that have more than one TP. In the rest of our discussion about TPs, we shall consider only those commit protocols that allow more than one TPs. The simplest such protocol is the one with |Q_0|=1, |N|=1, |Pc|=1, Pa=∅, |Com|=1 and |Ab|=1, none other than the three-phase commit protocol we have described in Section 3.3.2. N={*w*}, Pc={*p*}, Pa=∅, Com={c} and Ab={a} for this protocol. The FSA are synchronous within a

state. If any site in a component is in the initial state, that component can abort the transaction. Similarly, if any site is in a final state, the component can move into that final state. The two states that need special consideration are *w* and p since they are adjacent and $p \in Pc$ while $w \in N$.

In the context of partitioning, we shall only consider the three-phase commit protocol and its TPs. This can be justified as follows: a) It is the simplest commit protocol with more than one TP. In that sense, it is the canonical protocol in this class. We will later see that it has a rich variety of TPs. b) By the definition of TP, the only states that need special consideration are those in N that are adjacent to states from $Com \cup Pc$ and the states in Pc that are adjacent to states in N.

The following observations regarding TPs are immediate :

i): $state(C) \subseteq N \rightarrow f(C) \neq x$,

ii): $state(C) \subseteq Pc \rightarrow f(C) \neq y$, for any TP f.

Informally, $state(C) \subseteq N$ implies that none of the sites in site(C) are aware of the global decision on the transaction. Thus, there is always a possibility that a site in another group has already aborted. Hence, no TP can map C to x. On the other hand, $state(C) \subseteq Pc$ implies that all the sites know the decision to be commit. In this case, a site in another group could have already committed so that f(C) cannot be y for any TP f.

We shall use these properties of TPs in the next two chapters and derive TPs optimal under certain useful measures of

performance. The basic frame work introduced in this chapter will serve as the foundation on which we build several interesting consequences there.

Chapter 6 : Optimal Termination Protocols for Partitioning - I

6.1. Introduction

6.2. Performance measures

6.3. Decentralized termination protocols

6.3,1 Component optimal DTPs

6.3.2 Site optimal DTPs

6.4. Centralized termination protocols

6.4.1 A highly optimal LTP

6.4.2 CTP vs DTP

6.1. Introduction

In this chapter, we define certain simple measures for the performance of termination protocols (TPs) of the three-phase commit protocol in the context of partitioning. These measures do not make use of any statistical information on partitioning but realistically represent the performance aspects of interest. They are independent of any specific environmental characteristics. In the next chapter, we shall prove that no statistical information is of any significant consequence in determining most of the optimal TPs. Here, we shall prove the interesting result that, 'for any TP, there corresponds a partition in which all except at most one component are mapped onto w'. Hence, every TP has a worst case partition in which almost all components wait. We concentrate on deriving TPs that

have good average case performance. Since groups represent the maximal subsets of sites that can inter-communicate, it is important that as many such groups as possible are active in spite of a partitioning, to process new transactions that potentially access the databases at a number of sites. On the other hand, when many transactions access only local databases, the number of active sites determines the availability of the system. Thus, we consider the number of waiting components and the number of waiting sites per partition to be the basic factors that a desirable TP should minimize simultaneously if possible.

In what follows, we first define the measures for the performance of TPs and then develop an optimality theory for the decentralized and centralized versions.

6.2. Performance measures

We define two measures for the performance of TPs. The first one measures the number of components that would be left waiting. The other measure represents the behaviour in terms of the number of waiting sites.

Definition 6.1: Let P be a commit protocol. A TP, f, is *component optimal* in a class of TPs F iff $|\{C \in D$ such that $f(C)=w\}|$ is minimum in F.

Definition 6.2: A TP f is *site optimal* in F iff $\sum_{f(C)=w} |C|$ is minimum in F.

In the rest of our discussion, we deal only with the three-phase protocol and do not explicitely mention the commit protocol unless it is different from the three-phase commit.

6.3. Decentralized Termination Protocols

Here, we define the decentralized TPs, derive several properties regarding our measures and obtain the optimality results.

Definition 6.3: A commit protocol is *decentralized* iff all the FSA are identical and each site communicates with all other sites.

The TPs of decentralized protocols will be referred to as decentralized termination protocols or DTPs in what follows. The following lemma presents a necessary condition for a DTP.

Lemma 6.1. Let f be a DTP. Let $C,C' \in D$ such that $state(C \cup C') \cap \{c,a\} = \emptyset$. Then f(C)=x and f(C')=y implies that $site(C) \cap site(C') \neq \emptyset$.

Proof: Assume that the claim is false and there are C,C' satisfying the above conditions. By Observations i and ii of Section 5.5.1, state(C)={p} and state(C')={w}. This, together with the hypothesis that $site(C) \cap site(C') = \emptyset$ means that $C \cup C' \in D$, thus violating the consistency property of f and leading to a contradiction. □

We shall see later that a Centralized TP need not satisfy this condition. From the definition of TP, this condition is also sufficient for any TP. Hence we have

Theorem 6.1: The necessary and sufficient conditions that f:D→{x,y,w} is a DTP are

i) f satisfies the preservation property, and,

ii) $state(C \cup C') \cap (Com \cup Ab) = \emptyset$, f(C)=x & f(C')=y implies that

site(C)∩site(C')≠∅. □

Two simple but powerful properties of DTPs which follow from Theorem 6.1 and Observations i and ii of Section 5.5 play a prominent role in the remainder of this chapter.

Property 1. Let C∈D such that state(C)={p}. Construct C' such that state(C')={*w*} and site(C')⊆ I-site(C) (complement of site(C)). Then for any DTP f, either f(C)=w or f(C')=w or both.

Proof : Assume that f(C)≠w and f(C')≠w. Since state(C)={p}, f(C)≠y by Observation (i). Hence, f(C)=x. Since state(C')={*w*}, f(C')≠x and hence, f(C')=y. But, since site(C)∩site(C')=∅, C∪C'∈D, not possible by Lemma 1. □

Property 2. Let C,C'∈D such that site(C)=site(C'), state(C)={*w*},state(C')={p}, f(C)=y and f(C')=x. Then, for any C'' such that site(C'')⊆ I-site(C), and state(C'')⊆{p,w}, f(C'')=w.

Proof : Assume f(C'')≠w, and, specifically that f(C'')=x. Since site(C'')∩site(C)=∅, C∪C''∈D as long as state(C'')⊆{p,*w*}. But, f(C)=y by the hypothesis. Similarly, we can show that f(C'') cannot be y either. □

6.3.1 Component Optimal DTPs:

To appreciate the arguments to be presented in Lemma 6.2 and some more results to follow, we first introduce an informal, intuitive representation for the TPs. Let us consider a simple

example of 3 sites 1,2,3 and list down all possible groups and components C with these sites such that state(C)$\subseteq${p,w}:

Groups : (1), (2), (3), (1,2), (1,3), (2,3).

Components : ((1,w)), ((1,p)), ((2,w)), ((2,p)), ((3,w)), ((3,p)), ((1,w),(2,w)), ((1,w),(2,p)), ((1,p),(2,w)), ((1,p),(2,p)), ((1,w),(3,w)), ((1,w),(3,p)), ((1,p),(3,w)), ((1,p),(3,p)), ((2,w),(3,w)), ((2,w),(3,p)), ((2,p),(3,w)), ((2,p),(3,p)).

Now, we rewrite the components in a simple, tabular fashion as follows :

sites :	1	2	3
1.	-	-	w
2.	-	w	-
3.	w	-	-
4.	-	w	w
5.	w	-	w
6.	w	w	-
7.	-	-	p
8.	-	p	-
9.	p	-	-
10.	-	p	p
11.	p	-	p
12.	p	p	-
13.	-	w	p
14.	-	p	w
15.	w	p	-

16. p w -

17. w - p

18. p - w

The first row in this table represents the component ((3,w)), second row represents the component ((2,w)) and so on. A TP is a function mapping the rows of this table to the three possible actions. Observation (i) implies that any row between 7 and 12 (both inclusive) is mapped to one of x, w. Observation (ii) implies that any row between 1 and 6 is mapped to one of y, w.

Now, let us consider the rows 1 and 12. These correspond to the components ((3,w)) and ((1,p),(2,p)). Clearly, these two components are concurrent. Thus, at least one of them should be mapped to w under any DTP. (1 can be mapped to y or w and 12 can be mapped to x or w. If 1 is mapped to y, then 12 cannot be mapped to x. If 12 is mapped to x, then 1 cannot be mapped to y.) Similar property can be seen to hold for the rows 2,11; 3,10; 4,9; 5,8; and 6,7. Thus, the rows from 1 to 12 are effectively partitioned into disjoint pairs such that at least one in each pair is mapped to w by any DTP. The number of such pairs is clearly 2^3-2. For any number of sites n, it can be seen to be two less than the number of possible binary strings of length n (two less because the strings with all zeros and all one's are not considered - all zeros correspond to the case of empty group and all one's to the case of no partitioning, both of no interest.)

We can now formally prove the lower bound as follows :

Lemma 6.2. Let f be a DTP. Then $|\{C\epsilon D|f(C)=w\}| \geq 2^n-2$, where n is the total number of sites.

Proof: Let $C\epsilon D$ such that state(C)={p}. Let $C'\epsilon D$ such that site(C')$\subseteq$I-site(C) and state(C')={w}. From Property 1 above, either f(C)=w or f(C')=w. Clearly, for any $C\epsilon D$ such that state(C)={p} & $|C|\leq n-1$, there is C' such that either f(C)=w or f(C')=w. The total number of $C\epsilon D$ such that state(C)={p} & $|C|\leq n-1$ is 2^n-2. Thus,

$$|\{C|f(C)=w\}|\geq 2^n-2 \ . \ \square$$

Now we define a class of DTPs from which we produce the optimal DTPs. The DTPs in this class are characterized by a pair of nonnegative integers a, c such that $a+c\geq n+1$ (where n is the total number of sites).

Definition 6.4: A *quorum-based function* characterized by a, c is a function $f:D\rightarrow\{x,y,w\}$ such that i) f has preservation property, ii) state(C)$\cap$Pc$\neq\emptyset$ & $|C|\geq c$ implies f(C)=x, iii) for any C that does not satisfy the conditions of (ii), if state(C)$\cap$N$\neq\emptyset$ & $|C|\geq a$ then f(C)=y and iv) f(C)=w otherwise.

Let $C,C'\epsilon D$ and f(C)=x and f(C')=y under a quorum-based function f. By the definition of f, $|C|\geq c$ and $|C'|\geq a$. Since $c+a>n$, this implies that site(C)$\cap$site(C')$\neq\emptyset$. Thus, f has the consistency property. f has the preservation property by the definition. Hence, any quorum-based function is a TP.

It is interesting to note that

Theorem 6.2. Let f be any DTP. Then there exists a partition of the network in which all components except at most one wait.

Proof: Clearly, this is true for any quorum-based TP. Using the Property 1 of DTPs, a partition can always be constructed with all components except at most one mapped to w for the non-quorum-based TPs. □

We prove that the quorum-based protocol with a=1 and c=n, is component optimal among all DTPs.

Lemma 6.3. Let g be the quorum-based TP with a=1 and c=n. Then $|\{C|g(C)=w\}|=2^n-2$.

Proof : From the definition of g, g(C)=w iff state(C)={p} & $|C|\leq n-1$. Hence, there are exactly $\sum_{k=1}^{n-1}\binom{n}{k}=2^n-2$ such C in D. □

Thus we have,

Theorem 6.3. Let P be a decentralized commit protocol. Then there exists a quorum-based TP which is component optimal. □

Referring back to our example, we see that the only components mapped to w by g are between the rows 7 to 12, both inclusive. All other rows, since they contain at least one site in state w, are mapped to y. Now, it is immediate to see that the roles of x and y are interchangeable, ie., we can safely map the rows 1-6 to w and the rest to x. This DTP is no other than the quorum-based TP with c=1 and a=n which can also be shown to be component optimal.

But, not all quorum-based TPs are component optimal. In general, for the quorum-based TP f given by c=k and a=m, assuming k≥m, we have, $|\{C|f(C)=w\}|=\sum_{h=1}^{m-1} 2^h \binom{n}{h} + \sum_{h=m}^{k-1} \binom{n}{h}$. The first term in this expression gives the total number of sets C such that $|C| \leq m-1$ while the second term gives the number of sets C such that $m \leq |C| \leq k-1$ and state(C)={p}. For instance, for f with c=n-2 and a=3, $|\{C|f(C)=w\}|=2^n-2+(n^2-n)$ and increases with the increasing values of a until a=n/2. Then it starts decreasing so that it attains the least value for a=n. At the same time, we can notice that, $|\{C|f(C)=w\}|=2^n-2$ if c=n-1 and a=2 . By symmetry, the DTP with a=n-1 and c=2 is also component optimal.

Not all component optimal TPs are quorum-based as well. We can produce a protocol *in between* two optimal quorum-based TPs, with c=n, a=1 and c=n-1 and a=2, which is not quorum-based, as follows : Consider a specific site i, map the set {(i,w)} onto w and C' with site(C')=I-{i} and state(C')={p} onto x, while all other sets are mapped as with the quorum-based TP with a=1, c=n. Looking back at our example, let 1 be the designated site. Then, the rows mapped to w by this DTP are 3,7,8,9,11 and 12. The row 10 is mapped to x and all others are mapped to y. Clearly this is component optimal and is not quorum-based. The construction of this protocol suggests that there are a number of such non-quorum-based DTPs which are component optimal. DTPs of this kind can be called for when certain sites are found to have high availabilty (or, low failure rate). Then, no component involving those sites will be mapped to w, if possible. But, depending on the number of such sites, the corresponding DTP may

or may not remain component optimal.

6.3.2 Site optimal DTPs: First, let us note that there are component optimal TPs that are not site optimal. For $n \geq 9$, the quorum-based TP with $a=3$ and $c=n-2$ is better in the site measure than the one with $a=2$ and $c=n-1$ which in turn, is always better than the one with $a=1$ and $c=n$. As an example, let $n=9$. Let the above DTPs be respectively denoted by f_1, f_2 and f_3. We can check that $\left(\sum_{f_2(C)=\omega} |C| - \sum_{f_1(C)=\omega} |C| \right) = 36$ and $\left(\sum_{f_3(C)=\omega} |C| - \sum_{f_2(C)=\omega} |C| \right) = 63$ in this case. This is an interesting observation since it leads us towards an optimal TP by suggesting that we might keep increasing the value of a and decreasing the value of c to reduce the total number of waiting sites. But obviously we cannot make $a=n$ and $c=1$. Thus there is a point where the number of waiting sites is minimized.

In general, the total number of waiting sites for the quorum-based TP with $c=k$ and $a=m$ (with the assumption that $k \geq m$) is given by $\sum_{r=1}^{m-1} r \, 2^r \binom{n}{r} + \sum_{r=m}^{k-1} r \binom{n}{r}$. The derivation of this expression is straight forward : consider the corresponding expression for the total number of waiting components and introduce a new factor which represents the sizes of the components. The values of c and a that minimize this expression will give us the optimal TP among the quorum-based TPs. But we shall proceed through a more intuitive path to find these values. As an example, let us start with the quorum-based TP with $a=2$ and $c=n-1$ and see how to improve upon it. Under this TP, all components of size n-1 in state p are mapped to x, all components

of size ≥2 in state *w* are mapped to y and no component with states from both is mapped to w. Thus, any component of size ≤n-2 which has all sites in p is mapped to w. Now consider a component C of size n-2 with all sites in p. Let us map C to x. By performing the mapping, we reduce the total number of waiting sites by n-2; but, owing to Property 2, also increase them by $2(2^2-1)=6$. Thus, if n≥9, the net effect is a reduction in the total number of waiting sites. By the same argument, we see that we can furthur reduce the total number of waiting sites if we map *all* components of size n-2 with all sites in p to x. Here, the gain is $(n-2)\binom{n}{n-2}$ while the loss is $2\binom{n}{2}(2^2-1)$. We generalize this argument to find the optimal point.

Let k_0 be the smallest number k such that $k \geq (n-k)(2^{n-k}-1)$. Denote by u the quorum-based TP with $a=k_0$ and $c=n-k_0+1$ and by q the one with $a=n-k_0+1$ and $c=k_0$. Clearly the total number of waiting sites is the same for the two TPs. This development shows that they are site optimal among all quorum-based TPs. In fact, we can further show that these are also optimal among *all* DTPs.

Lemma 6.4. Let f be any DTP. Then there exists a quorum-based TP s such that

$$\sum_{s(C)=\omega} |C| \leq \sum_{f(C)=\omega} |C| .$$

Proof:

Case 1: There are no C,C' such that site(C)=site(C'), state(C)={*w*}, state(C')={p}, f(C)=y and f(C')=x.

This implies that for any subset M of sites I, either U or V, where site(U)=site(V)=M, state(U)={p}, state(V)={w}, is mapped to w. Since there are $\binom{n}{r}$ subsets of sites with size r, for any r, this implies that $\sum_{f(C)=\omega} |C| \geq \sum_{r=1}^{n-1} r \binom{n}{r}$.

Consider the quorum-based TP s with a=2 and c=n-1.

$\sum_{f(C)=\omega} |C| = 2n + \sum_{r=2}^{n-2} r \binom{n}{r} \leq \sum_{r=1}^{n-1} r \binom{n}{r}$

implies $\sum_{f(C)=\omega} |C| \leq \sum_{f(C)=\omega} |C|$.

Case 2: There exist C,C' such that site(C)=site(C'), state(C)={w}, state(C')={p}, f(C)=y and f(C')=x.

First, we look at an example. Let n=9. The smallest m such that $m \geq (n-m)(2^{n-m}-1)$ is 7. Let us denote by ES(m) the total number of sites waiting under the quorum-based TP with c=m and a=n-m+1. Then, we can calculate that ES(8) - ES(7) = 36 and ES(6) - ES(7) = 1260. (We have used the general expressions for ES under any quorum-based TP.)

Formally, first we prove that the TP with c=m and a=n-m+1 is site optimal among the quorum-based TPs, by showing that ES(m) ≤ ES(m+1) and ES(m) < ES(m-1), for the above m and that it is the only value for which these inequalities hold. Thus, we are showing that it is a point where the ES has the minimum value over the range [n/2,n-1] for c. In the other range [1,n/2-1], we will have a similar DTP, owing to the symmetry in c and a.

Writing the expressions for ES(m), ES(m-1) and ES(m+1), we can see that,

$ES(m+1) - ES(m) = (n - (n-m)2^{n-m})\binom{n}{m}$. But, this is

≥ 0, by the definition of m.

$ES(m-1) - ES(m) = ((n-m+1)2^{n-m+1} - n)\binom{n}{m-1}$.

But, since m is the smallest with the given property, $(m-1) < (n-m+1)(2^{n-m+1}-1)$. It follows that $ES(m-1) - ES(m) > 0$.

Conversely, we can show that the smallest m with the given property is the only value of c in the range $[n/2, n-1]$ such that $ES(m)<ES(m+1)$ and $ES(m)<ES(m-1)$. Hence, it is optimal among the quorum-based TPs.

Assume that f is a non-quorum-based TP.

case a). Assume that, for any group S, there is a C with site(C)=S and f(C)=w. Then, clearly, $ES(g) \leq ES(f)$.

case b). Assume that there are groups S such that for all C with site(C)=S, f(C) is x or y. Let S be the smallest such group. If $|S| \leq \lfloor n/2 \rfloor$, then it is easy to see that $ES(g) < ES(f)$. Assume that $|S| > \lfloor n/2 \rfloor$. Now, we show that $ES(f) \geq ES(u)$.

Let T be the set of S satisfying the above property. If $|S'| > k_o$ for some S' not in T, then pick any **maximal** group S' not in T and put it into T. Clearly, this does not increase the ES(f), due to the definition of k_o. Thus, in general, all groups S' not in T, with sizes larger than k_o, can be placed into T without incresing ES(f), so that f coincides with u(q). On the other hand, if $|S| < k_o$ for some S in T (and $|S| > \lfloor n/2 \rfloor$), then pick the **minimal** such S in T and delete it from T. Again, due to the definition of k_o, this cannot increase ES(f). Repeating this process for all such S, f coincides with u or q. □

Theorem 6.4. There are at most two quorum-based TPs which are site optimal and these are the only site optimal TPs. □

Cor. 6.1: For n≥9, no component optimal TP can be site optimal and vice versa.

Proof: We know that $ES(m)=\sum_{r=1}^{m-1} r.2^r\binom{n}{r} + \sum_{r=m}^{n-m} r\binom{n}{r}$. In the proof of Lemma 6.4, we have derived the expression $ES(m-1) - ES(m) = ((n-m+1)\ 2^{n-m+1}-n)\ \binom{n}{m-1}$. This is nonnegative only if $(n-m+1)2^{n-m+1} > n$. For n≥9, this is possible only when m≤n-2. □

6.4. Centralized Termination Protocols

Here, we present a slightly generalized version of centralized protocols. We first need a partial ordering on the states of FSA as : t≤s iff the distance of s to its nearest final state is no more than the distance of t to its nearest final state. We say a set of sites L is a *leading set* iff for C∈D, (i,s),(j,t)∈C and i∈L implies t≤s.

Definition 6.5: A commit protocol is *centralized* iff it has a leading set and the sites in the leading set can communicate with all other sites while the non-leading sites communicate only with the leading sites.

We call the TPs of a centralized commit protocol *Centralized Termination Protocols*(CTPs). Since most of the existing commit protocols are centralized, this class is of great practical importance. In this section, we study the termination protocols for centralized commit protocols. We prove that the performance of certain centralized TPs is better than *any* DTP, thus providing

a theoretical justification for the use of centralized commit and termination protocols in the context of network partitioning.

First we should note that one can also use DTPs for terminating centralized commit protocols. In this sense, DTPs form a proper subset of CTPs. This is not surprising since the *commit property* of TPs holds even if the function is restricted over a proper subset of its domain while the *preservation property* is satisfied by all TPs. Next we observe that there is a *natural* class of centralized TPs which is most interesting. These TPs try to exploit the leading set property of the centralized commit protocol in an explicit fashion. Observe that, if a leading site is in w, all sites in the network are in states from $\{w,q_c\}$. We call these TPs *leading set TPs* or LTPs. While looking for CTPs optimal in component and site measures, we deal only with LTPs and restrictions of DTPs.

Definition 6.6: A CTP f is an LTP iff for all $C \epsilon D$, $site(C) \cap L \neq \emptyset$ implies $f(C) \neq w$ where L denotes the leading set.

Cor. 2. In absence of the simultaneous failure of all leading sites, there is a component of any partition that can successfully terminate under an LTP.

Though it is possible to abort a transaction in all cases in which the transaction is not yet committed and it is known that no other component can commit it, we consider this approach to be very conservative. Instead, we insist that a transaction should be committed if it is possible to do so consistently. Clearly, this cannot degrade the performance of an LTP. Thus, we assume

that, for any LTP f, there is $C \epsilon D$ such that $state(C) \subseteq \{p,w\}$ & $f(C)=x$.

Lemma 6.5. *Let f be an LTP. Then,* $site(C) \cap L=\emptyset$ & $state(C)=\{w\}$ implies that $f(C)=w$.

Proof: Note that, if there are C', C'' such that $C' \cup C \epsilon D$, $C'' \cup C \epsilon D$, $(site(C') \cup site(C'')) \cap site(C)=\emptyset$, $f(C')=x$, and $f(C'')=y$, then $f(C)=w$ since f is a TP. It is easy to see that one can produce C', C'' with the above property since f is an LTP and $site(C) \cap L=\emptyset$. For instance, take $site(C'')=site(C')=L$, $state(C'')=\{w\}$ and $state(C')=\{p\}$. □

6.4.1 A highly optimal LTP

Based on the definition of LTP and the above Lemma, we can construct an LTP h algorithmically as follows:

i) If $state(C) \cap (Com \cup \{p\}) \neq \emptyset$, then $h(C)=x$.
ii) If $state(C) \cap (Ab \cup Q_o) \neq \emptyset$, then $h(C)=y$.
iii) If $site(C) \cap L \neq \emptyset$, then $h(C)=y$; iv) $h(C)=w$ otherwise.

It is easy to see that h satisfies the preservation and commit properties. Thus, it is an LTP. We can also observe that h does not satisfy the conditions specified in Lemma 6.1, that is, $h(C)=x$ and $h(C')=y$ does not necessarily imply that $site(C) \cap site(C') \neq \emptyset$. In the following, we show that h is also *somewhat* quorum-based and is highly optimal among the LTPs.

Definition 6.7: A quorum-based TP f is *weighted* if, in the definition of the quorum-based TP, each site is assigned a weight and the size of a set is replaced by its weight, i.e., sum of the

weights of its constituent sites.

Observe that, by assigning weights to sites as $\omega(i)=n$ if $i \epsilon L$ and 1 otherwise, h coincides with the weighted quorum-based TP with $c=1$ and $a=n$. We can furthur show that h is the 'best' among the LTPs in the following strong sense :

Theorem 6.5. Let f be any LTP. Then, $h(C)=w$ implies $f(C)=w$, for any $C \epsilon D$.

Proof: By definition, $h(C)=w$ implies that $\omega(C) \leq n-1$ & $state(C)=\{w\}$. Thus, $f(C)=w$ for any LTP f, by Lemma 6.5. □

It is now straight forward to show that h is component and site optimal among the LTPs :

Theorem 6.6. The weighted quorum-based TP h is both component and site optimal among the LTPs.

Proof: Since $h(C)=w$ implies $f(C)=w$ for any LTP f and $C \epsilon D$, $|\{C \mid h(C)=w\}| \leq |\{C \mid f(C)=w\}|$. By the same reason, $\sum_{h(C):\omega} |C| \leq \sum_{f(C):\omega} |C|$. □

Thus, we have produced a CTP which is both site and component optimal among the the 'proper' CTPs. It still remains to see if the same CTP is better than the restrictions of DTPs that can be used in conjunction with the centralized commit protocols.

6.4.2 **CTP vs DTP:** In the following section, we shall prove that the weighted quorum based TP h performs better than any

restricted DTP in both component and site measures. First, let us note that the optimal DTPs reported in Lemmas 3 and 4 remain optimal even after restricting their domains to the centralized case. (The proof of Lemma 6.3 remains unchanged while the arguments given in the proof of Lemma 6.4 remain valid.)

Theorem 6.7. Let f be any restricted DTP. Then,

$$\sum_{h(C)=\omega} |C| < \sum_{f(C)=\omega} |C| \quad \text{for sufficiently large n.}$$

Proof : It has been shown that the quorum-based TP q, defined by $a=n-r+1$ and $c=r$ where r is the smallest integer such that $r \geq (n-r).(2^{n-r}-1)$, is site optimal among all DTPs. Thus, $\sum_{q(C)=\omega} |C| \leq \sum_{f(C)=\omega} |C|$ for all DTPs f. Now it remains to be shown that $\sum_{h(C)=\omega} |C| < \sum_{q(C)=\omega} |C|$, thus proving that h is better than any DTP. By definition, q(C)=w implies that either i) $|C| \leq n-r$ or ii) state(C)={p} & $n-r<|C|<r$. Note that, due to the restriction of the domain, the total number of waiting sites is slightly less than that in the unrestricted case. In the unrestricted case, $\sum_{q(C)=\omega} |C| = \sum_{k=1}^{n-r} k\, 2^k \binom{n}{k} + \sum_{k=n-r+1}^{r-1} k \binom{n}{k}$. The first term in this expression is different in the restricted case since, if a site in L is in w, no other site can be in p state. It is easy to see that the following inequality holds in this case:

$$\sum_{q(C)=\omega} |C| \geq \sum_{k=1}^{n-r} k\, 2^{k-|L|} \binom{n}{k} + \sum_{k=n-r+1}^{r-1} k \binom{n}{k}$$

$$= \sum_{k=1}^{n-1} k \binom{n}{k} + \sum_{k=1}^{n-r} k\,(2^{k-|L|}-1)\binom{n}{k} - \sum_{k=1}^{n-r} (k+r-1)\binom{n}{k+r-1}.$$

On the other hand, $\sum_{h(C)=\omega} |C| = \sum_{k=1}^{n-|L|} k \binom{n-|L|}{k}$, since $c=1$. For $|L| \geq 1$, $\sum_{h(C)=\omega} |C| \leq \sum_{k=1}^{n-1} k \binom{n-1}{k}$. For sufficiently large n, $\sum_{h(C)=\omega} |C| < \sum_{q(C)=\omega} |C|$.

Theorem 6.8. Let f be defined as in the preceding theorem. Then,

$$|\{C \epsilon D \mid h(C)=w\}| < |\{C \epsilon D \mid f(C)=w\}|.$$

Proof: $|\{C \mid h(C)=w\}| = \sum_{k=1}^{n-|L|} \binom{n-|L|}{k}$, by the definition of h.

For $|L| \geq 1$, $|\{C \mid h(C)=w\}| \leq \sum_{k=1}^{n-1} \binom{n-1}{k} < \sum_{k=1}^{n-1} \binom{n}{k} \leq |\{C \mid f(C)=w\}|$ by Lemma 6.2 . □

These results clearly show that, *good* protocols handling network partitioning are, after all, not as expensive as they first appeared to be. This is particularly so, because, one needs to invoke a termination protocol for network partitioning only when a partitioning is detected, which is quite infrequent.

Chapter 7 : Optimal Termination Protocols for Partitioning - II

Contents

7.1. Introduction

7.2. Analysis of the commit protocols

7.3. Partitions and probabilities

7.4. Measures for TPs

7.5. Decentralized TPs

7.5.1 Component optimality

7.5.2 Site optimality

7.6. Centralized TPs

7.7. Selection of the coordinator

7.8. Conclusion

7.1. Introduction

In the previous chapter, we have studied the termination protocols and derived TPs optimal under two important measures - component and site optimality measures. These measures, as defined there, have implicitely assumed that - a) the probabilities of occurrence of all possible components are identical, and, b) the probability of finding a site in state w is the same as that of finding it in state p at any point of time. We can see that these assumptions do not hold in certain realistic environments. In this chapter, we generalize our

measures of TPs by considering the probabilities of occurrence of different partitions to be potentially different and the probabilities of finding sites in w and p states to be possibly different. We first analyze the decentralized and centralized versions of the three-phase commit protocol and derive the probabilities of various components. We then present our notions of partitions, and the probabilities associated with them. We generalize our measures to take all the available information into account and develop the optimality theory in the centralized and decentralized cases.

Several results in this chapter have their counterparts in Chapter 6. Hence, we do not present the detailed proofs here unless they are appreciably different.

7.2. Analysis of the commit protocol

Decentralized three-phase commit. (Fig. 1)

Phase 1: A site in state q_0 receives a transaction request, sends the subtransactions to the participating sites and moves into the state 'w'; each participating site in q_0 decides whether the subtransaction can be committed or not. If the subtransaction is to be aborted, the site aborts the subtransaction, sends a 'no' message to all other participating sites and moves into a final state 'a'. Otherwise, it sends an 'yes' and moves into a state 'w'.

Phase 2: Each site receives messages from all other sites. If any of them is 'no', it aborts, sends an 'abort' message to all others and moves into 'a'. Otherwise, it sends a 'prepare to commit' message and moves into 'p'.

Phase 3: Each site commits the subtransaction and moves into the final state 'c' if 'prepare to commit' messages are received from all the sites.

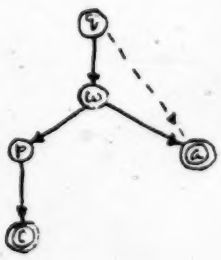
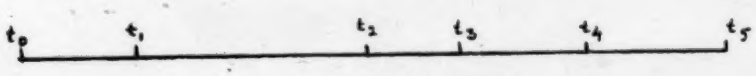


Fig. 1: The FSA corresponding to the decentralized commit

A typical transaction execution, based on the decentralized protocol, can be depicted as follows on a global time scale, assuming the decision to be *commit*:



$[t_0,t_1)$: The sites receive the request

$[t_1,t_2)$: all participating sites are in w and none of them has received the 'yes' messages from all other sites

$[t_2,t_3)$: the sites move from w into p after receiving all 'yes' messages

$[t_3,t_4)$: all sites are in p, but none has received the 'prepare to commit' messages from all other sites

$[t_4,t_5)$: the sites move from p into c

Network partitioning assumption: A partitioning can occur with the same probability at any point of time during a transaction execution.

Clearly, this is a very natural assumption since there is no a priori way of knowing when a partitioning occurs.

Assume a partitioning occurs at time t. Consider a group S of size k and let P(m,k) be the sum of the probabilities of all components of S with exactly m sites in state p, for $0 \leq m \leq k$. The reason for considering the sum of these probabilities rather than the individual probabilities is two-fold: a) Since we are dealing with the decentralized protocol where all sites have the same

status under the protocol, the probability of being in a particular state is the same for all sites, and, b) we can derive the probability for a component independent of the components it can concurrently occur with.

From the above time-scale representation, we see that, for $0<m<k$,

$$P(m,k) = \binom{k}{m} \frac{1}{(t_s-t_0)} \int_{t_2}^{t_3} \left(\frac{t-t_2}{t_3-t_2}\right)^m \left(\frac{t_3-t}{t_3-t_2}\right)^{k-m} dt$$

Repeatedly integrating by parts, we obtain,

$$P(m,k) = \frac{1}{(k+1)} \frac{(t_3-t_2)}{(t_s-t_0)} .$$ (Please see Appendix for the details)

Since the value $P(m,k)$ depends only on k, we observe that the probabilities are the same for all m. For $m=0$ and $m=k$, the following expressions can be similarly derived:

$$P(k,k) = P(m,k) + \frac{(t_4-t_3)}{(t_s-t_0)}, \text{ and,}$$

$$P(0,k) = P(m,k) + \frac{(t_2-t_1)}{(t_s-t_0)}, \text{ for } 0<m<k.$$

We present an example network and derive the probabilities of occurrence of the components.

Assume that there are 4 sites in the network and consider a partition $R = \{(1,3),(2,4)\}$. We assume the commit protocol to be the decentralized three-phase protocol. For simplicity, we assume that $(t_2-t_1)/(t_s-t_0) = (t_3-t_2)/(t_s-t_0) = (t_4-t_3)/(t_s-t_0)$.

Let z denote this value. The list of possible components is as follows:

sites :	1 3 2 4	1 3 2 4
states :	w w w w	p w w w
	w w w p	p w w p
	w w p w	p w p w
	w w p p	p w p p
	w p w w	p p w w
	w p w p	p p w p
	w p p w	p p p w
	w p p p	p p p p

Looking at all the four sites as a single group, we can find that $P(1,4) = P(2,4) = P(3,4) = (1/5)z$ and $P(0,4) = P(4,4) = (6/5)z$.

The following observations can be made from the above formaulae, which can be easily checked with the example :
a) Since $(t_2-t_1)=(t_4-t_3)$, we have $P(0,k)=P(k,k)$ for any k, and,
b) If (t_2-t_1) and (t_4-t_3) are of the order of (t_3-t_2), then, $P(0,k)$, $P(k,k)$ are of the order of $k.P(m,k)$. But, it can be seen from the protocol that $(t_3-t_2)>>(t_2-t_1)\cong(t_4-t_3)$, so that, $P(0,k)\cong P(k,k)\cong P(m,k)$. (Nevertheless, this *does not* mean that all components for a partition have the same probabilities of occurrence, as we will see in Section 7.3.)

Now we consider the centralized version of this commit protocol (Fig. 2) and derive similar relationships:

Phase 1: The central site (in q_0) receives a transaction and sends the subtransactions to the participating sites, moving into w; each participating site (in q_0) receives its

subtransaction. Each site decides whether the subtransaction can be committed or not. If the subtransaction cannot be committed, it is aborted (by moving into 'a') and the site sends a 'no' to the central site. Otherwise, an 'yes' is sent (by moving into w).

Phase 2: The central site receives all the messages. If one of them is 'no', it aborts (moving into 'a') and sends 'abort' to all participating sites. Otherwise, it sends 'prepare to commit' (and moves into p). Each site receives the message, aborts if the message is 'abort'(moving into 'a'); otherwise it moves into p and sends an 'ack' to the central site.

Phase 3: After receiving all 'ack's, the central site sends out 'commit' messages(moving into 'c'). Each site commits (by moving into 'c') after receiving 'commit'.

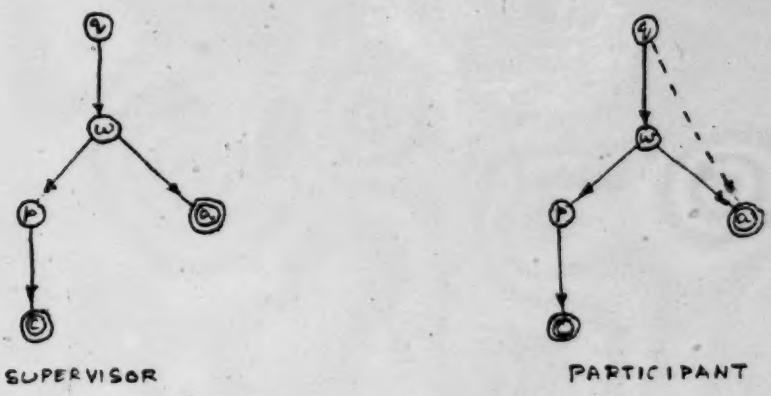
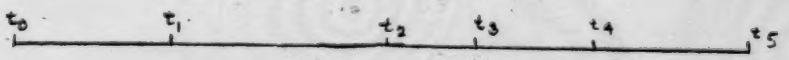


Fig.2: The FSA corresponding to the centralized commit

Let us look at a typical execution of the centralized version of the commit protocol on the global time scale:



$[t_0,t_1)$: sites receive the subtransaction

$[t_1,t_2)$: all participating sites are in w and the central site has not received all the messages from the sites.

$[t_2,t_3)$: sites move from w into p

$[t_3,t_4)$: all sites are in p

$[t_4,t_5)$: sites move from p into c

Let us consider any subset S of sites that does not contain the central site. Then the analysis is exactly the same as in the decentralized case. If S contains the central site, *no other site can be in p unless the central site is in p.* Assuming that the local processing times are small, in this case, we have $(t_2-t_1)=(t_4-t_3)$, and, $(t_3-t_2)=(t_1-t_0)=(t_5-t_4)$. We have the following expressions when the set of sites under consideration contains the central site.

$$P(m,k) = \frac{(t_3-t_2)}{k(t_5-t_0)}, \text{ for } 0<m<k,$$

$$\frac{(t_2-t_1)}{(t_5-t_0)} < P(0,k) < P(m,k) + \frac{(t_2-t_1)}{(t_5-t_0)}, \text{ and,}$$

$$P(k,k) \cong P(0,k) + P(m,k).$$

7.3. Partitions and probabilities

Let I be the set of sites in the network. A network *partition* R can be represented as $R = \{S_1, S_2, .., S_s\}$ where $S_i, S_j \subseteq I$, $S_i \cap S_j = \emptyset$ for all $0<i<j\leq s$ and $\cup S_i = I$. Since the sites can be in different states, a partition can be realized as different collections of components.

A physical partitioning does not depend on the states of the transactions at the sites. Hence, it is reasonable to assume

that one can specify the probability of occurrence of a particular partition independent of the states, denoted as P(R). Assume that the set of possible partitions,R, is known. Let Groups(R), Comp-set(R) respectively be the set of all groups and the set of all components for the given collection of partitions. For any $C \epsilon$ Comp-set(R), let k=|C| and P(C) the probability of occurrence of C. Then, clearly, $P(C)=(\sum P(R)$ where $site(C) \epsilon R) * \beta(C)$ while $\beta(C)$ may be either P(o,k), $\frac{P(m,k)}{\binom{k}{m}}$ or P(k,k) depending on the protocol and on state(C).

7.4. Measures for TPs

Now, we generalize the measures to incorporate the statistical information.

Let f be a TP and $T(R,f)= \{C \epsilon$ Comp-set(R)$|\ f(C)=w\}$.

Definition 7.1. A TP f is said to be *component optimal* in a set of TPs F if and only if $EC(R,f)=(\sum P(C)$ for $C \epsilon T(R,f))$ is minimum in F.

Definition 7.2. A TP f is *site optimal* in F if and only if $ES(R,f)=(\sum P(C)|C|$ for $C \epsilon T(R,f))$ is minimum in F.

These measures can be seen to coincide with the previous measures when P(C)=P(C') for all $C,C' \epsilon D$. Observe that we have incorporated all the available information into these generalized measures.

Let us restate two important properties of TPs (given in Chapter 6) that will be of great importance in developing the

optimality results. We shall present them here in a form most appropriate in this context :

Property 1 : Let S, S' be two groups such that $S \cap S' = \emptyset$. Let f be a TP. Let site(C) = S and state(C) = {w} and site(C') = S' and state(C') = {p} for $C, C' \in D$. Then, either f(C)=w or f(C')=w or both.

Property 2 : Let f be a TP. Let site(C) = site(C') = S, state(C) ={w} and state(C') = {p} for a group S and $C, C' \in D$. Assume $f(C) \neq w$ and $f(C') \neq w$. Let S' be any other group such that $S \cap S' = \emptyset$ and C'' any component with group S'. Then, f(C'')=w.

7.5. Decentralized TPs

Definition 7.3. Let R be a partition and C,C' components of R. We say they are *complementary* if and only if i) site(C)=site(C') and ii)state(C)={w} and state(C')={p}.

For any TP f and any collection R of partitions, we define

$U(R,f) = \{S \in Groups(R) \mid f(C) \neq w$ and $f(C') \neq w$ for the complementary components C,C' with site(C)=S$\}$,

$V(R,f) = \{S \in Groups(R) \mid f(C)=f(C')=w$ for the complementary components C,C' with site(C)=S$\}$,

$W(R,f) = Groups(R) - (U(R,f) \cup V(R,f))$.

We can observe that, if $S \cap S' = \emptyset$ and $S \in U(R,f)$, then S' must be in V(R,f) for any f. Similarly, if $S \in W(R,f)$, then $S' \in V(R,f) \cup W(R,f)$.

Let us look at a simple example here: Let I={1,2,3,4,5,6}, R={R_1,R_2} where R_1={{1,2,3},{4,5,6}} and R_2={{1,2,3,4},{5,6}}. Consider the DTP f defined as follows:

i) f has preservation property,

ii) if site(C)={1,2,3} and pϵstate(C), then f(C)=x,

iii) if site(C)={1,2,3} and p$\notin$state(C), then f(C)=y,

iv) if site(C)={1,2,3,4} and pϵstate(C), then f(C)=x, and,

v) f(C)=w otherwise.

Groups(R)={{1,2,3},{1,2,3,4},{5,6},{4,5,6}} and the only group such that f(C)$\neq$w and f(C')$\neq$w for its complementary components C,C' is {1,2,3}. Thus, U(R,f)={{1,2,3}}. For the group {1,2,3,4}, the component C with state(C)={p} is mapped to x and its complementary component is mapped to w. Thus, {1,2,3,4}ϵW(R,f). The other two groups are in V(R,f) since all components of these groups are mapped to w.

7.5.1 Component optimality

Here also, we can gain great insights through a tabular representation similar to the one in Chapter 6. As an example, consider 4 sites, 1,2,3,4. Consider a partition involving two groups, (1,2) and (3,4). Let us write down the components formed by these two groups individually :

sites :	1	2		3	4
1.	w	w	5.	w	w
2.	w	p	6.	w	p
3.	p	w	7.	p	w

4. p p 8. p p

It is evident that the rows 1 and 8 cannot both be mapped to y and x respectively. Same is true for 5 and 4 also. If 1 is mapped to y and 4 is mapped to x, then all the components between 5 and 8 are mapped to w. Same is true with 5 and 8. But, by mapping 1 and 5 to y or 4 and 8 to x, only two components are mapped to w. Clearly, this is the best any DTP can do. This argument can be extended for any number of sites, leading to a lower bound on the waiting components in the following theorems :

For a partition R, let All-w(R)={C|state(C)={w}, site(C)ϵR} and All-p(R)={C|state(C)={p}, site(C)ϵR}. Thus, for any CϵAll-p(R), state(C)={p}, and, state(C)={w} for any CϵAll-w(R).

Theorem 7.1. Let f be a DTP and R be a partition. Assume $\exists$ C,C'ϵAll-w(R) such that f(C)$\neq$w and f(C')$\neq$w. Then, f(C'')=w for all C''ϵAll-p(R). It is true when we interchange the roles of All-w(R) and All-p(R) also.

Proof: Assume $\exists$ C,C'ϵAll-w(R) such that f(C)$\neq$w and f(C')$\neq$w. f(C)$\neq$w implies that $\forall$C''ϵD such that site(C'')$\cap$site(C)=$\emptyset$ and state(C'')={p}, f(C'')=w. That is, f(C'')=w for all C''ϵAll-p(R) - {C'''} where site(C''')=site(C). Similarly, f(C')$\neq$w implies that f(C'')=w for all C''ϵAll-p(R) - {C'''} where site(C''')=site(C'). Hence, f(C)$\neq$w $\wedge$ f(C')$\neq$w implies that f(C'')=w for all C''ϵAll-p(R). The converse can also be similarly proved. $\square$

This result directly leads to a powerful lower bound for the component measure :

Theorem 7.2. Let f be a DTP and R any partition. Then, $EC(\{R\},f) \geq \min\{\sum P(C) \text{ for } C \in All\text{-}p(R), \sum P(C) \text{ for } C \in All\text{-}w(R)\}$.

Proof : We have seen in the above proof that if any one component in All-p(R) or All-w(R) is mapped into {x,y}, then at least (|R|-1) components are mapped to w. The above theorem states that if two components from any one of these two sets are mapped into {x,y}, then |R| components are mapped to w. Thus, the best any DTP can do is to map *all* components in one of those sets into {x,y}. □

This result can be extended to any collection of partitions trivially as follows :

Cor. 7.1 : Let f be a DTP and R any collection of partitions. Then, $EC(R,f) \geq \min\{\sum_{R \in R} (\sum P(C) \text{ for } C \in All\text{-}p(R)), \sum_{R \in R} (\sum P(C) \text{ for } C \in All\text{-}w(R))\}$. □

Let us denote the expression on the right hand side of the above inequality by LB(R). Our first lead to the optimal DTP comes from the following corollary in which we show that no DTP f such that $U(R,f) \neq \emptyset$ can achieve the above lower bound :

Cor. 7.2 : Let R be a collection of partitions and f a DTP such that $U(R,f) \neq \emptyset$. Then, $EC(R,f) > LB(R)$.

Proof: Since $U(R,f) \neq \emptyset$, there exist complementary components C, C' such that $f(C) \neq w$ and $f(C') \neq w$. By Property 2, $f(C'') = w$ for any

component C'' such that site(C'')∩site(C)=∅. Furthermore, recall that the sum of the probabilities of all components of a given group S, within a partition R, is 1.

Let R∈R such that R∩U(R,f)≠∅. Then, EC({R},f)≥P(R)(|R|-1) by the above two observations. But, LB({R}) < P(R).|R|/2 since min{P(0,k),P(k,k)} < 1/2 for any k. Thus, EC({R},f) > LB({R}) in this case. For R∈R such that R∩U(R,f)=∅, EC({R},f)≥LB({R}) by Theorem 7.2. Thus, EC(R,f) > LB(R). □

In the above proof, we have considered the individual partitions and shown our result to be true for each of them. This is a very interesting property since it shows that our results are independent of the collection of partitions. We shall see that this technique can be used in proving virtually all the results to come. Thus, it is no surprise that we would find *fixed* TPs to be optimal.

The above result shows that W(R,f)=Groups(R) for any component optimal DTP. We have already seen a DTP with this property in Chapter 6. We redefine it here and call it g :

Assume that P(k,k)≤P(0,k) for any k.

i) g satisfies the preservation property, and,

ii) *w*∈state(C) implies g(C)=y; otherwise, g(C)=w '.

It is obvious that U(R,g)=∅ and W(R,g)=Groups(R) for any R. Now, we show that g achieves the lower bound of Cor. 7.1.

'If P(k,k)>P(0,k), then we can make g(C)=x whenever p∈state(C) and g(C)=w when state(C)={*w*}.

Theorem 7.3. g is a component optimal DTP for any R.

Proof : By the definition of g, g(C)=w only if state(C)={p}. Hence, for any RϵR, EC({R},g)=$\sum$P(C) for CϵAll-p(R). Since this is true for any R, g is component optimal for any R. □

Observe that the Cor. 7.2 above eliminates all but two DTPs as candidates for being component optimal. One of them is g defined above and the other is its complementary function which maps any component with one site in p to x and all others to w. Both of these DTPs are component optimal if the values of P(0,k) and P(k,k) are same for all k.

Thus, we have shown that the decentralized termination protocol g leads to the minimum expected number of waiting components independent of the set of possible partitions and their probabilities of occurrence. This is a very interesting result since it amounts to saying that this TP can be effectively used in any environment. Furthermore, it does not depend on any of the assumptions we have made regarding the probabilities of the components.

7.5.2 Site optimality

Recall that we have an approximate equality between P(0,k) and P(m,k) for any k and m, under the decentralized commit protocol.

Theorem 7.4. Assume that there is a positive integer K such that no group in Groups(R) intersects with more than K groups in

Groups(R). Then, for sufficiently large n, g is site optimal among the DTPs.

Proof: If $U(R,f)=\emptyset$ for some f, then it is straight-forward to see that $ES(f) \geq ES(g)$.

Suppose that $U(R,f)\neq\emptyset$ for some f. By definition,

$$ES(f) \geq \sum_{R \in \mathbf{R}} P(R) * \sum_{S \in R \cap W(R,f)} |S| * P(0, |S|) + \sum_{R \in \mathbf{R}} P(R) * \sum_{S \in R \cap V(R,f)} |S| .$$

On the other hand,

$$ES(g) = \sum_{R \in \mathbf{R}} P(R) * \sum_{S \in R} |S| * P(0, |S|).$$

Since $P(0,k)=P(m,k)$, $P(0,k) \equiv 1/k$ for all groups of size k for any k. Thus, clearly, ES(g) is bounded above by $2 * \sum_{R \in \mathbf{R}} P(R) * |R|$.

Now, we show that $ES(f) > 2 \sum_{R \in \mathbf{R}} P(R) * |R|$ for sufficiently large n. A reasonably large lower bound on the number of groups in V(R,f) is all that we need for this purpose. By the hypothesis, no group intersects with more than K other groups. Hence, there are at least (n-K-1) groups which are in V(R,f) when a group S is in U(R,f). Thus, $ES(f) > (n-K-1) \sum P(R)$. For large n, the size of any partition R, $|R| \ll n$ and for realistic networks, more the number of groups in a partition, less is the probability for that partition. Hence, $ES(g) \leq ES(f)$ for sufficiently large n. □

Though the assumption of bounded intersection is reasonable in practice, it is still important to consider the general case. But, there does not seem to be a polynomial time solution to the problem of finding the site optimal DTP in this general case. We could neither show that it is NP_Complete. The basic problem one faces is that a bound on the number of groups in any one of the three sets U(R,f), V(R,f) or W(R,f) seems to be hard to obtain. In fact, even finding if one of them is empty seems to be difficult.

We conjucture that the problem of finding if any one of these sets is empty is NP-Complete. If this is true, then the original problem is also clearly NP-Complete.

7.6. Centralized TPs

As we have seen in Chapter 6, a DTP can also be used in conjunction with a centralized commit protocol, though it takes no advantage of the existence of the central site. Here, we see that the component optimal DTP g performs well in this new role also.

Assume that i is the central site in the discussion to follow.

Theorem 7.5. Let f be a DTP used in conjunction with the centralized commit protocol. Assume that $U(R,f)\neq\emptyset$ and $i \notin S$ for all $S \in U(R,f)$. Then, $EC(R,f) > EC(R,g)$.

Proof: Observe that none of the DTPs take advantage of the existence of a central site. Thus, effectively, there is no difference in the relative performances of f and g as CTPs or as DTPs. Hence, the proof follows on the same lines as Cor. 7.2 □

Since P(k,k) is larger than P(0,k) for a component with the central site, let g map any component with a site in p to x and any component with all sites in *w* to w. The following result can be proved similar to Theorem 7.4 :

Theorem 7.6 Let f be as above. Assume that no more than half of the partitions have groups of size ≥ n-2. Then, $ES(R,f) \geq$

ES(R,g). □

In the above two theorems, we have compared g only with the other DTPs. We can even prove a stronger result about g here. The proof is straight forward.

Theorem 7.7. Let f be any CTP such that $U(R,f)=\emptyset$. Then, $EC(R,f)\geq EC(R,g)$ and $ES(R,f)\geq ES(R,g)$. □

Now, let us consider the CTPs f such that $U(R,f)\neq\emptyset$. Among them, first consider the subclass for which $S\in U(R,f)$ implies $i\notin S$. These are very similar to the DTPs and it is easy to argue, as in Theorem 7.4, that g performs better than them under the same assumptions.

Theorem 7.8. Let f be a CTP such that $U(R,f)\neq\emptyset$ and $S\in U(R,f)$ implies $i\notin S$. Then, $EC(R,f)\geq EC(R,g)$ and $ES(R,f)\geq ES(R,g)$. □

Thus, g is better than many CTPs as well in both the component and site measures. Clearly, we have not exhausted all CTPs in our characterization. Specifically, one case remaining is of CTP f for which $U(R,f)\neq\emptyset$ and $i\in S$ for some $S\in U(R,f)$. We shall now define one such CTP which we prove to be optimal in both components and sites. This CTP tries to take advantage of the central site's existence.

Define h as follows: for any $C\in$Comp-set(R), a) $h(C)=x$ if $p\in$state(C), b) $h(C)=y$ if $i\in$site(C) $\wedge$ state(C)={w}, and, c) $h(C)=w$ otherwise.

Here, $U(R,h)\neq\emptyset$ and $S\in U(R,h)$ implies $i\in S$.

This is the same as the highly optimal LTP of Chapter 6. As in there, we can show it to be optimal among all CTPs. But first, we show it optimal in the subclass of CTPs f for which $U(R,f)\neq\emptyset$ and $i\epsilon S$ for some $S\epsilon U(R,f)$. Observe that the arguments of Theorem 7.4 are not valid in this case since the Property 2 we are appealing to is not applicable for $S\epsilon U(R,f)$ such that $i\epsilon S$. Assuming f to be such that it does not wait if it has the central site, as in Chapter 6, the following strong result is immediate :

Theorem 7.9. Let f be a CTP such that $U(R,f)\neq\emptyset$ and $i\epsilon S$ for some $S\epsilon U(R,f)$. Then, $h(C)=w$ implies $f(C)=w$. □

We have explicitly divided the class of TPs that can be used in conjunction with the centralized protocol into several subclasses and shown that one of the two TPs g and h is optimal in each of them. Now, we compare the performances of g and h and show that h is indeed better than g :

Theorem 7.10. $EC(R,g) > EC(R,h)$.

Proof: Since h maps a component C to w only if $state(C)=\{w\}$ and $i\not\in site(C)$, $EC(\{R\},g)>EC(\{R\},h)$ for any partition $R\epsilon R$. □

Theorem 7.11. $ES(R,g) > ES(R,h)$.

Proof: Similar to Theorem 7.10. □

Thus, we have proved that the CTP h is optimal both in components and sites, and, this is true independent of the possible set of partitions R and the probabilities of occurrence of individual partitions. Intuitively, h seems to make use of all the available information in any component, simultaneously

ensuring the consistency of the databases.

7.7. Selection of the coordinator

We have shown that in the context of network partitioning, centralized protocols are more effective under certain practical measures. But, the effectiveness of centralized TPs depends on the central site also. For instance, if the central site occurs in very small groups in all the possible partitions, then the expected number of waiting sites can be larger than if it were in relatively large groups. Thus, we see that, even though h is site optimal for a given central site i, there can be a wide range of such h depending on which one of the sites is the central site. In this section, we address the problem of finding the optimal such h. By the above discussion, we see that it boils down to that of finding a site which maximizes the effect of the corresponding h function. We thus formulate the question as:

> find a site i such that, given the possible partitions and their probabilities of occurrence, choosing i as the central site makes the CTP h_i (the CTP h with i assumed to be the central site) superior to h_m for any $m \neq i$ in the sense that $ES(R,h_i) \leq ES(R,h_m)$.

Here, we derive the conditions under which h_i is superior to h_m under the site optimality measure.

Theorem 7.12. $ES(R,h_i) \leq ES(R;h_m)$ if and only if

$$\sum_{\substack{S \in GROUPS(R), \\ i \in S}} [P(0,|S|).|S|(\sum_{S \subseteq R} P(R))] \geq [\sum_{\substack{m \in S \\ S \in GROUPS(R)}} P(0,|S|).|S|(\sum_{S \subseteq R} P(R))].$$

Proof: From the definition of h, we see that

$$ES(R,h_i) = \sum_{i \notin S} P(0,|S|).|S|(\sum_{S \in R} P(R))$$

$$= \sum_{S} P(0,|S|).|S|(\sum_{S \in R} P(R)) -$$

$$\sum_{i \in S} P(0,|S|).|S|(\sum_{S \in R} P(R)).$$

Similarly, we can express $ES(R,h_m)$ as well. Our claim follows directly from these expressions. □

The above conditions are rather simple and very useful since we can now find a simple algorithm which can be used to find the optimal h :

For each group S in the collection, find $W(S) = (P(0,|S|).|S|(P(R)))$. For site j, let $W(j) = (\Sigma W(S))$ where $j \in S$. Find i such that W(i) is the maximum. Designate i as the central site.

Time complexity of this algorithm is clearly linear in the input size, making it optimal in time.

7.8.Conclusion

In Chapter 6, we have concluded that there is a CTP whose performance is superior to any DTP in most practical situations. The fact that this is so even under the most general measures, as shown in this chapter, strengthens our conviction that it is desirable to employ centralized commit protocols when highly reliable sites exist in a network. A very surprising outcome of our studies is that the statistical information on the probable partitions is of little consequence in designing the optimal TPs. This is of great importance since it implies that we can safely

employ the same protocol in any environment.

Chapter 8 : Conclusions

Contents

8.1. Introduction

8.2. Resiliency of Distributed databases - the past

8.3. Contributions of this thesis

8.4. Dealing with the other transaction models

8.4.1 Hierarchical model

8.4.2 Nested model

8.5. Suggested future directions

8.5.1 Site failure

8.5.2 Network Partitioning

8.1. Introduction

In this Chapter, we present in a nutshell our contributions in enhancing the understanding of fault-tolerance in distributed database systems. We demonstrate the state of the art in this area and discuss the possible directions the future research could take.

8.2. Resiliency of Distributed databases - the past

Consistency constraints in databases reflect the physiacl environment being modelled by the database. Thus, it is

imperative that any user accessing the database sees a consistent database. Atomic transactions are provided as the means by which a user can access a database, so that, by definition, each transaction leaves a consistent database consistent. Thus, the only two possible ways a transaction can be completed are a) by executing it completely and incorporating all its effects permanently into the database, and, b) by making sure that none of its effects are incorporated into the database. These two actions are known as *commit* and *abort* respectively. Atomicity of a transaction in a distributed environment thus amounts to either a) committing at all sites at which the transaction is executed , or, b) aborting at all sites.

The fundamental requirement for any Transaction Management System is that it implements all transactions only as atomic actions. Thus, the basic problem we are concerned with is, *How to ensure that any distributed transaction is atomically completed in spite of failures in the system ?* Several possible failures have been considered in the literature. Our interest lies in two of them - site failures and network partitioning.

In addition to guaranteeing the atomicity, we require that as many operational sites in the distributed system as possible are allowed to complete the transactions incomplete at the time of a failure.

Alsberg and Day [AD 76] were the first to explicitly consider the problem of site failures in the context of transaction atomicity. They have considered the centralized

model of decision and proposed simple protocols to ensure that the system is resilient to upto k arbitrary site failures for any fixed k (k-resiliency). Gray has considered the decentralized model of decision where participating sites can unilaterally abort in the early stages of the protocol [Gray 78]. He has proposed and analyzed the well-known two-phase commit protocol. Several variations of this protocol are studied in the literature, all of them turning out to be 'blocking' to arbitrary site failures [Gray 78, HS 80]. Certain attempts are made to make the two-phase protocol nonblocking to arbitrary site failures, leading to the discovery that it is essential to add one more phase to that protocol [Garcia 79, Skeen 81].

The problem of network partitioning has not been explicitely addressed until recently. Certain researchers, who have tried to solve the concurrency control problem together with the transaction atomicity, have proposed consensus-based approaches which automatically guarantee atomicity in presence of site failures and network partitioning [BL 82, Gifford 79, LB 82, Thomas 79, Skeen 82a]. In such schemes, all the operational sites wait when no consensus has been reached at. All the existing database systems have ignored the partitioning problem.

The first formal treatment of fault-tolerance in distributed databases is reported in a thesis by Skeen [Skeen 82], who has exclusively considered the decentralized model of decision. He has modelled protocols through finite state automata and defined the concepts of commit, termination and recovery protocols. He

has introduced the notions of blocking and nonblocking protocols and derived the necessary and sufficient conditions for nonblocking commit protocols for arbitrary site failures. A class of commit protocols nonblocking to site failures, known as three-phase protocols are also introduced. The nonexistence of commit protocols nonblocking to partitioning has ben proved and a class of protocols similar to consensus-based protocols have been proposed to deal with partitioning.

Cooper [Cooper 82] has shown that the existing commit protocols do not deal satisfactorily with the partitioning problem. He has taken a probabilistic approach, considering the local transaction processing time a random variable. The basic assumption he makes is that the transmission delays are negligibly smaller than the local processing times.

8.3. Contributions of this thesis

In the context of site failures, we have asked the following questions :

a) What are the conditions necessary and sufficient for nonblocking, under the centralized decision model ?
b) How do the results change if the 'arbitrary site failures' are restricted to a bounded number of simultaneous failures ?
c) What are the possible strategies for site recovery from failures ? How are they related to the characteristics of the commit and termination protocols being used ?

d) How can the concept of backup sites be modelled ?

e) How can we take advantage of special properties of transactions ?

f) Are there formal models conceptually more attractive than the existing one ?

Since site failures can be considered as specialized partitioning, we have defined a general failure environment encompassing both these failures. We have proposed a new information-based model to study the commit protocols. We have used the first-order predicate logic as the tool for working in this model and rederived several existing results. Under the centralized decision model, which was not formally studied earlier, we have proved that all delayed-commit protocols are nonblocking, thus showing that there is a wealth of nonblocking protocols in this model. At the same time, we have shown that none of the protocols under centralized model have any effect in dealing with the partitioning problem. For this purpose, we have introduced the notion of non-trivial termination protocols which capture the essential features of a protocol effective for partitioning.

In the decentralized decision model, we have derived the Fundamental relation among the failures, which can be stated as, *The nonblocking problem for site failures is equivalent to the non-trivial termination problem of partitioning*. This theorem thus presents the subproblem of partitioning which has exactly the same complexity as the nonblocking problem of site failures.

In fact, their equivalence is in a very strong sense : solution of one problem is also a solution of the other. Thus, the nonblocking problem of site failures is the *same* as the non-trivial termination problem of partitioning. This result is the first theoritical demonstration of the relation among complexities of these two problems.

We have introduced the notion of k-bounded failure where not more than k sites can simultaneously fail, for a predetermined fixed value k. We have shown how the concept of backup sites naturally fits into this model. We have derived results on the nonblocking aspects showing that such protocols require relatively small number of message exchanges. We have investigated the effects of this new model on the termination and recovery aspects of commit protocols. For termination, we have proved that the number of messages is far less than in the arbitrary failure model. In fact, we need a constant number of messages in most realistic situations. We have furthur shown that it is true for both centralized and decentralized termination protocols.

We have investigated the question of how special properties of transactions can be exploited. Specifically, we have studied the read-only transactions and derived simple protocols nonblocking to both site failures and partitioning. Here, we have showed how hierarchical and nested transaction models also can have inexpensive protocols for read-only transactions.

Regarding site recovery, we have classified the possible site recovery strategies depending on the 'ease' with which a site can recover. The recovery aspects depend on both the commit and termination protocols used. We have defined three possible strategies - independent recovery, recovery after successful communication with any participating site, and, recovery after successful communication with a designated site, in the descending order of their desirability. We have shown that the centralized decision model is far inferior to the decentralized model in the recovery aspects. In general, we have shown that there is a natural trade-off in the termination and recovery aspects, for a given commit protocol. In particular, we have shown that, if a nonblocking TP is used, then no site is allowed to recover independently. Even otherwise, it is shown that atmost one site can recover independently. The decentralized decision model is shown to allow the second recovery strategy for all sites, while the centralized model allows such recovery to atmost one site, the central site.

Our major thrust of investigation has been on the network partitioning problem where the existing literature is very sparse. Since there is no nonblocking commit protocol, we have explored several possible means of achieving the best possible performance in presence of partitioning. The fundamental assumption we have made in this context is that the database consistency cannot be violated, even temporarily. This implies that no two operational sites can complete a transaction in two different modes (one by committing and the other by aborting).

We have looked for commit and termination protocols optimal under certain realistic measures for the performance in presence of partitioning. We have defined the TPs as functions mapping 'components', that is, the groups together with all the information in that group, onto the possible decisions. This model of a TP has been particularly useful in our pursuit for 'best' TPs to deal with partitioning. We have considered two specific measures for TPs - expected number of waiting components and the expected number of waiting sites. They measure the availability of the distributed database system after the partitioning but before the complete restoration.

First, we have assumed that all possible partitions are equally probable and the probabilities of finding any site in one the two 'critical' states are the same. With these assumtions, our measures of expected behaviour have been expressed as simple formulas. We have then studied both the decentralized and centralized versions of the TPs to produce TPs optimal under the measures. Some of the important results we have obtained are -

a) There are simple quorum-based decentralized TPs optimal under the measures,

b) No decentralized TP optimal in one measure is optimal in the other,

c) There is a centralized TP optimal in both the measures, and,

d) The optimal centralized TP performs better than any decentralized TP in both the measures.

In the domain of centralized TPs, the results c) and d) are very positive since they imply that, a) centralized TPs are more effective than decentralized TPs, and, b) a single TP can optimize both the measures, the property we are looking for.

We have then generalized our working environment to consider different partitions to have possibly different probabilities of occurrence. Furthermore, we have analyzed the decentralized and centralized versions of the three-phase commit protocol and derived the probabilities of finding sites in the 'critical' states. We have shown that the probabilities of finding sites in the w,p states play a predominent role in determining the optimal TPs. In fact, we have proved the surprising result that the optimality theory is independent of the statistical information available. The consequences of this result are far reaching - a) the TPs derived can be utilyzed with equal effect in any realistic environment, b) there is no need to collect any statistical information for designing useful TPs, and, c) these TPs, suitably modified, can be used in conjunction with any commit protocol having more than one TP, etc., We have also proved that there is a DTP optimal in both components and sites in this general model.

We have obtained similar results for the centralized commit protocol : a) there is a single TP optimal in both the component and site measures, and, b) this TP is independent of the possible partitions. We have shown that the problem of finding the site optimal centralized TP is naturally related to the problem of

selecting a central site. The optimal algorithm we have presented for this central site selection problem can be used to designate more than one central sites or to designate backups to a central site.

8.4. Dealing with the other transaction models :

Most of the results in this thesis assumed the relaxed central-control model of transactions. We have claimed that these results can be extended to the other models easily. Here, we briefly discuss how this can be done.

8.4.1 Hierarchical model

The subtransactions of any (sub)transaction look upon their parent exactly similar to the coordinator in the central-control model. The parent also looks at its children as the participating sites in that model. Hence, the conglomerate of a parent and its children is exactly similar to a central site and the participating sites in the central-control model.

Given a commit protocol P for the centrally controlled model, we can derive a protocol P' for the hierarchical model as follows:

Transaction-initiation : Each parent acts as the central site in P except that it does not prepare all the commands to be followed by the descendents. It only sends out the commands to its immediate descendents.

Reach-decision : The parent receives the decisions of its

childeren, passes on to its parent. The root node makes the global decision from its children's decisions.

Transaction-completion : The transaction-completion rule of P is followed by each parent and its children. A parent waits for messages from its parent before it sends to its children.

All the existence results of Chapters 3,4 and 5 can be extended into the domain of tree transactions. The optimality results of Chapter 6 also hold in principle, but, the probabilistic treatment of Chapter 7 requires modifications to incorporate the larger delays involved in changing from one state to the other. We do not study this case, but leave it as an open problem.

8.4.2 Nested model

The structure of a nested transaction is very similar to that of a tree transaction, except that there is synchronization among the subtransactions of a transaction in this model. A subtransaction, after finding that all its immediate descendents have committed, can commit with no more message exchanges. It can make an abort decision if one of them have aborted. (Let us note that there is a vital difference between a commit by subtransactions in this model and the hierarchical model : here, the commit is only temporary - it is not visible to the (sub)transactions external to the given transaction hierarchy. Thus, the parent of a committed subtransaction can later abort

it. Finally, the whole hierarchy is either committed or aborted.) Its parent is then notified of this completion and proceeds similarly. Thus, the protocols required for nested transactions are more like the protocols for read-only transactions and our discussions about those protocols hold here. We leave it an open question to be studied.

8.5. Suggested future extensions

It is not clear how the optimality results for partitioning in the general environment (where the various probabilities can be derived) change for the hierarchical model of transaction. For the nested transactions, one needs to see how the new assumptions that a) a subtransaction can permanently commit even if its parent aborts, and, b) a parent can commit even if certain designated subtranscations abort, effect the consistency aspects. These two problems require immediate attention.

The nested transaction model is currently the most general model for distributed transactions. One can think of more general models, for instance, those that have the interactions among the subtransactions in the form of general (acyclic) digraphs. The semantics of such transaction models need to be worked out. This is an interesting direction to take to see if such models lead to better resiliency of distributed databases.

Recently, there have been some attempts to integrate the site failure problem with other kinds of failures not considered here [MSF 83]. Such an integrated approach might lead to more

insights into the nature of these failures. Particularly, one might be able to derive interesting relations among the failures, similar to our Fundamental relation among site failures and partitioning.

8.5.1 Site failures

We have seen that there are certain interesting problems directly related to the various site recovery strategies. One of them is deciding how long a site needs to store the decision on completed transactions to ensure that it can pass that information to any querying site, to consistently terminate the transactions. Several strategies may be possible to minimize the amount of time a completed transaction is remembered. Similarly, if a site recovering from failure needs to communicate with all the central sites, strategies are to be worked out that minimize the time taken for such a recovery.

8.5.2 Network partitioning

Our approach has been to disallow any inconsistencies into the database, even temporarily. Another possible approach is to abandon this restriction and to resolve any inconsistencies at the time of group merging [Davidson 82, Wright 83]. This is known as the 'optimistic' approach since it assumes that there wil be very few transactions that need to be backed out during the merging. After a partitioning, the sites behave as if there were no failure and go ahead with new transactions. When a merging takes place, each site first prepares a serial schedule

of the transaction actions executed since the partitioning has occurred and all the sites in the new group try to merge their schedules into a serial schedule. In the process, certain transactions may have to be backed out. This approach has led to certain theoritically interesting problems. There are several open problems that need to be looked into.

Site recovery (rather, 'group recovery') in the context of partitioning is equivalent to merging two groups. No theory has been developed to deal with this problem when we insist that consistency cannot be violated. Specifically, the problem of finding an optimal merging algorithm is open and requires immediate attention. It is not a straight forward situation due to the possibility that there could be a reconfiguration while two groups are being merged. The problem becomes involved if sites 'forget' that they have already given their consent for a decision in one group and then moved into another group. The simplest situation is when sites maintain a history of partitioning as known to them and make use of it while making their later moves.

In Chapter 7, we have assumed that transmission delays are comparable to the local processing times. The site optimality results we have obtained do not remain valid if we assume that the transmission delays are negligibly small (or, the transaction processing times are relatively very large). In fact, the problem of determining the site optimal TPs seems to be NP-Hard in this case. Though it may not be of any practical

significance, it remains open to see how this new assumption effects the various results.

References

[ABG 81] R. Attar, P.A. Bernstein, and, N. Goodman, "Site Initialization, Recovery and Back-up in a Distributed Database System," TR-13-81, Aiken Computation Laboratory, Harvard University.

[AD 76] P.A. Alsberg, and J.D. Day, "A Principle for Resilient Sharing of Distributed Resources," Proc. of 2nd Symposium on Software Engineering, 1976, pp. 562-570.

[Balter 81] R. Balter, "Selection of a Commitment and Recovery Mechanism for a Distributed Transactional System," Proc. of IEEE Conf. on Reliability in Distributed Software and Database Systems, 1981, pp. 21-26.

[BG 81] P.A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," ACM Computing Surveys, June 81, pp. 185-222.

[BG 82] P.A. Bernstein and N. Goodman, "A Sophisticate's Introduction to Distributed Database Concurrency Control," 8 th Int'l Conf. on VLDB, Sept. 82, pp. 62-76.

[Bhargava 82] B. Bhargava, "Resiliency Features of the Optimistic Concurrency Control Approach for Distributed Database Systems," Proc. of IEEE Conf. on Reliability in Distributed Software and Database Systems, 1982, pp. 19-32.

[BL 82] H. Breitweiser and M. Leszak, "A Distributed Transaction Processing Protocol Based on Majority Consensus," ACM SIGACT-SIGOPS Conf. on PODC, Aug. 82, pp. 224-237.

[Cooper 82] E.C. Cooper, "Analysis of Distributed Commit Protocols," ACM SIGMOD 82, pp. 175-183.

[Davidson 82] S. Davidson, "An Optimistic Protocol for Partitioned Distributed Databases," Ph.D. Thesis, Princeton University.

[DG 81] S.B. Davidson and H. Garcia-Molina, "Protocols for Partitioned Distributed Database Systems," Proc. of IEEE Conf. on Reliability in Distributed Software and Database Systems, 1981, pp. 145-149.

[DR 82] D. Dolev and R. Reischuk, "Bounds on Information Exchange for Byzantine Agreement," ACM SIGACT-SIGOPS Symposium on PODC, Aug. 82, pp. 132-140.

[DS 82]D. Dolev and H.R. Strong, "Requirements for Agreement in a Distributed System," Proc. 2nd Int'l Symposium on Distributed Databases, Sept. 82.

[DS 82a] D. Dolev and H.R. Strong, "Polynomial Algorithms for Multiple Processor Agreement," Proc. 14th ACM SIGACT SOTC, May 82.

[DS 82b] D. Dolev and H.R. Strong, "Distributed Commit with Bounded Waiting," Proc. 2nd Symposium on Reliability of

BLANK PAGE INSERTED

BLANK PAGE INSERTED

Distributed Software and Database Systems, July 82.

[DS 82c] D. Dolev and H.R. Strong, "Authenticated Algorithms for Byzantine Agreement," IBM RR RJ3416(82).

[EGLT 76] K.P. Eswaran et al. "The Notion of Consistency and Predicate Locks in a Database System," CACM, Nov. 76.

[Garcia 79] H. Garcia-Molina, "Performance of Update Algorithms for Replicated Data in a Distributed Database," Ph.D. Thesis, Princeton University.

[Garcia 80] H. Garcia-Molina, "Reliability Issues for Completely Replicated Distributed Databases," IEEE COMPCON, Fall 80, pp.442-449.

[Garcia 82] H. Garcia-Molina, "Elections in a Distributed Computing System," IEEE Tr. on Computers, Jan. 82, pp. 48-59.

[Gifford 79] D.K. Gifford, "Weighted Voting for Replicated Data," Proc. of 7th Symposium on Operating Systems Principles, pp. 150-162.

[GMBLLPPT 81] J. Gray et al. "The Recovery Manager of the System R Database Manager," ACM Computing Surveys, June 81, pp. 223-242.

[Gray 78] J.N. Gray, "Notes on Database Operating Systems," in Lecture Notes in C.S.,Operating Systems, an advanced course, Springer-Verlag, pp. 393-481.

[Gray 81] J.N. Gray, "The Transaction Concept : Virtues and Limitations," 7th Int'l Conf. on VLDB, 81, pp. 144-154.

[HS 80] M. Hammer and D. Shipman, "Reliability Mechanisms for SDD-1 : A System for Distributed Databases," ACM TODS, Dec. 80, pp. 431-466.

[Kohler 80] W.H. Kohler, "Overview of Synchronization and Recovery Problems in Distributed Databases," IEEE COMPCON, Fall 80, pp. 433-441.

[Lamport 78] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," CACM July 78, pp. 558-565.

[Lampson 78] B.W. Lampson, "Atomic Transactions," Chapter 11,Distributed Systems Architecture and Implementation, Springer-Verlag 105.

[LB 82] M. Leszak and H. Breitweiser, "A Fault-tolerant Scheme for Distributed Transaction Commitment," 3rd Int'l Conf. on Distributed Computing, Oct. 82.

[LSP 82] L. Lamport, R. Shos[illegible] and M. Pease, "The Byzantine Generals Problem," AC[illegible]PLS, July 82, pp. 382-401.

[ML 83] C. Mohan and B. Lindsay, "Efficient Commit Protocols for Tree of Processors Model of Distributed Transactions," Proc. 2nd ACM SIGACT-SIGOPS Symposium on PODC, Aug. 83.

[Moss 81] J.E.B. Moss, "Nested Transactions : An Approach to Reliable Distributed Computing," Ph.D. Thesis,MIT.

[Moss 82] J.E.B. Moss, "Nested Transactions and Reliable Distributed Computing," Proc. of IEEE Conf. on Reliability in Distributed Software and Database Systems, 1982, pp. 33-39.

[MPM 80] D.A. Menasce, G.J. Popek and R.R. Muntz, "A Locking Protocol for Resource Coordination in Distributed Databases," ACM TODS, June 80, pp. 103-138.

[MSF 83] C. Mohan, H.R. Strong and S. Finkelstein, "Method for Distributed Transaction Commit and Recovery using Byzantine Agreement within Clusters of Processors," Proc. 2nd ACM SIGACT-SIGOPS Symposium on PODC, Aug. 83.

[PSL 80] M. Pease, R. Shostak and L. Lamport, "Reaching Agreement in the Presence of Faults," JACM, April 80, pp. 228-234.

[Reed 83] D.P. Reed, "Inplementing Atomic Actions on Decentralized Data," ACM TOCS, Feb. 83, pp. 3-23.

[RS 82] D.R. Ries and G.C. Smith, "Nested Transactions in Distributed Systems," IEEE TSE, May 82, pp. 167-172.

[Skeen 81] D. Skeen, "Nonblocking Commit Protocols," ACM SIGMOD 81, pp. 133-142.

[Skeen 81a] D. Skeen, "A Decentralized Termination Protocol,"

Memo. No. UCB/ERL M81/50, U. of Calif., Berkeley.

[Skeen 82] D. Skeen, "Crash Recovery in Distributed Database Management System," Ph.D. Thesis, U. of Calif., Berkeley.

[Skeen 82a] D. Skeen, "A Quorum-based Commit Protocol," TR 82-483, Cornell University, Ithaca.

[SM 78] R.M. Schapiro and R.E. Millstein, "Failure Recovery in a Distributed Database System," IEEE COMPCON, Spring 78, pp. 66-69.

[SS 83] D. Skeen and M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System," IEEE TSE, May 83.

[TGGL 82] I.L. Traiger et al. "Transactions and Consistency In Distributed Database Systems," ACM TODS, Sept. 82, pp. 323-342.

[Thomas 79] R.H. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," ACM TODS, June 79, pp. 180-209.

[Wright 83] D.D. Wright, "On Merging Partitioned Databases," ACM SIGMOD 83, pp. 6-14.

APPENDIX

DERIVATION OF P(m,k) FOR ALL m.

Given a set of sites, we want to derive the probabilities that, a) exactly m of k sites are in state p and (k-m) are state w, b) all of them are in state p, and, c) all of them are in state w. Let us consider P(m,k) first, for $0 < m < k$. From the time-scale representation, it is clear that a network partitioning occuring at time t can find this situation only if $t_2 \leq t < t_3$. For any site, closer t is to t_3, higher is the probability that that site is in p. Thus, given that a site is in either w or p, and that some sites are in p, the probability that it is in p at time t is given by $(t - t_2)/(t_3 - t_2)$. Different sites can be in w or p independent of each other, so that the probability that exactly m sites are in p at t is given by

$$\left(\frac{t - t_2}{t_3 - t_2}\right)^m \left(\frac{t_3 - t}{t_3 - t_2}\right)^{(k-m)}$$

To compute the total probability corresponding to all such t, we need to integrate this expression between t_2 and t_3. The resultant expression is to be normalized since $(t_5 - t_0)$ need not be unity. Thus, we get,

$$P(m,k) = \binom{k}{m} \frac{1}{t_5 - t_0} \int_{t_2}^{t_3} \left(\frac{t-t_2}{t_3-t_2}\right)^m \left(\frac{t_3-t}{t_3-t_2}\right)^{k-m} dt$$

(We are multiplying by $\binom{k}{m}$ since exactly m out of k sites can be in p in those many possible ways.)

A repetetive application of integration by parts gives the simplified expression, $\frac{t_3-t_2}{t_5-t_0} \cdot \frac{1}{k+1}$

Let us consider P(k,k) now. All sites are in p during the interval $[t_3, t_4)$. Furthermore, all k sites we are interested in may have reached p before other sites. Thus, there may be a fraction of the interval (t_2, t_3) during which all these k sites are in p. Thus, we get,

$$P(k,k) = \frac{t_4-t_3}{t_5-t_0} + \frac{1}{k+1}\frac{t_3-t_2}{t_5-t_0}$$

P(0,k) can also be similarly given since all sites are in w during $[t_1, t_2)$ and the k sites may remain in w for slightly longer period.

CENTRALIZED PROTOCOL.

The case when the central site is not in the group is the same as in the decentralized case. Now, assume that the central site is among the k sites we are interested in. Then, for $0 < m < k$, the following expression can be given :

$$P(m,k) = \binom{k}{m}\frac{1}{t_5-t_0} \cdot \int_{t_2}^{t_3} \left(\frac{t-t_2}{t_3-t_2}\right)^{m-1} \left(\frac{t_3-t}{t_3-t_2}\right)^{k-m} dt$$

This is because we know that the central site moves into p at t_2.

The other two expressions can be derived similarly.